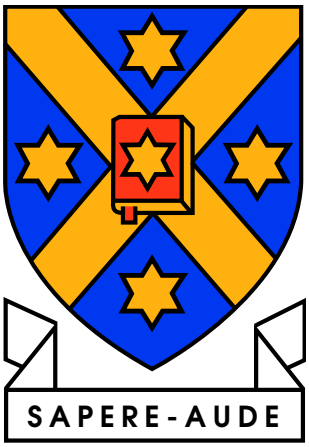
APPROXIMATING SOLUTIONS TO CONVECTION-DIFFUSION EQUATIONS BY TENSOR TRAIN DECOMPOSITIONS



Jandré Snyman
Supervised by Professor David Bryant and Associate Professor Colin Fox

A thesis submitted in fulfilment of the requirements of the degree of Master of Science

Department of Mathematics and Statistics
University of Otago

December, 2020

Abstract

A finite volume method for solving general time-homogeneous convection-diffusion equations with zero source term is presented. Computational efficiency of the method is improved by performing linear algebra in the tensor train format. To our knowledge this is the first time that the tensor train format and the finite volume method have been combined for this purpose.

Finite volume methods, tensors and tensor decompositions are reviewed by summarising prominent texts on each respective topic. We extend a finite volume method for convection equations to include diffusion terms and show that the method preserves integrals and positivity. The time discretisation uses an explicit Euler step that leads to a sequence of linear systems of equations defining a discrete approximation of the solution at some final time. The recurrence is stepped forwards in time by performing algebraic operations in the tensor train format. In some cases, this leads to a significant increase in computational efficiency.

We use our tensor train implementation of the finite volume method to approximate the allele frequency spectrum in three populations by solving the Wright-Fisher diffusion equation. Our method did not appear to outperform current methods for approximating the allele frequency spectrum. However, we develop some interesting and efficient tools for approximating the allele frequency spectrum if the solution to the Wright-Fisher diffusion equation is known in tensor train format.

Acknowledgements

First and foremost, I want to thank my supervisors David Bryant and Colin Fox. Thank you for your consistent support and guidance during the entire project. Your insightful feedback has pushed me to be a better researcher and mathematician. Most of all, thank you for introducing me to new and exciting topics in mathematics.

I would like to thank Ivan Oseledets and everyone involved with the TT-toolbox Matlab package. Without your algorithms and code many of the practical implementations in this thesis would not have been possible. Thank you to Ivan Oseledets and Dmitry Savostyanov for your prompt and helpful replies to my questions about your work.

I would also like to express my sincere gratitude to the University of Otago for giving me a generous scholarship to pursue this project.

Thank you to my fellow postgraduate students who shared an office with me this year. I appreciate all the times we discussed our projects with each other and the insights you all offered. Thanks especially to Josh Bromell for all the crash courses in genetics.

I cannot forget to thank my family and friends for all the unconditional support in this very intense year. In particular, I want to thank my wife and best friend Michaela. Thank you for putting up with me being sat at a desk for hours on end and thank you for being just as committed to my work as I am. Thank you for your unconditional love and support.

Contents

Chapter 1

Introduction

1.1 Motivation

The *allele frequency spectrum* (abbreviated AFS) is the distribution of derived allele frequencies in populations of the same species. The sensitivity of the AFS to demographic parameters such as population size and migration rates has led to its use for inferring these parameters ([18],[23], [33]). Assuming Wright-Fisher reproduction and an infinitely many sites model the time evolution of the AFS is well approximated by a diffusion process [7, Chap. 7].

The idea of diffusion processes was introduced to the field of population genetics by Fisher [11] and Wright ([65],[66]). Their ideas were significantly extended by Kimura [24] who showed in [25] that a certain density of allele frequencies in P populations satisfies a convection-diffusion equation of the form

$$\begin{aligned}\frac{\partial\phi}{\partial\tau}(\tau, x_1, x_2, \ldots, x_P) = &\sum_{p=1}^{P} \frac{\partial^2}{\partial x_p^2}\left[\phi(\tau, x_1, x_2, \ldots, x_p)\frac{x_p(1-x_p)}{2\nu_p}\right] \\ &- \sum_{p=1}^{P} \frac{\partial}{\partial x_p}\left[\phi(\tau, x_1, x_2, \ldots, x_P) \sum_{q=1,\ q\neq p}^{P} M_{p\leftarrow q}(x_q - x_p)\right], \qquad (1.1)\end{aligned}$$

on the bounded domain $(0,1)^P$, each x_p-axis denotes the relative frequencies of alleles in the pth population. The variables ν_p and $M_{p\leftarrow q}$ are constants depending on the specific evolutionary model. We call (1.1) the *Wright-Fisher diffusion equation*. Once the density ϕ is known at a specific time the expected AFS at that time is computed by an L^2 inner product with a P-dimensional continuous version of the binomial distribution function.

Although Kimura solved the Wright-Fisher diffusion equation for one population in his 1964 paper [25] so far, no analytic solutions have been found for multiple populations. Usually, in practice multiple populations need to be considered. For this, reason numerical approximations of the density ϕ are used instead. In [18] a numerical method based on finite differencing schemes is developed and optimised for two and three populations. Their method is implemented in a publicly available Python [64] package known as ∂a∂i. Another method developed in [34] uses truncated expansions of the solution in complete bases of a certain functional space. In [33] this method was shown to be computationally feasible for up to 4 populations. While the ∂a∂i software package is widely used in population genetics research, many applications require the Wright-Fisher diffusion equation

to be solved for a large number of populations. So, creating a method that is computationally efficient and runs in reasonable time for higher dimensions would be a sizeable contribution to the field.

The number of populations determines the dimension of the Wright-Fisher diffusion equation. So, for quadrature-based approximation methods such as ∂a∂i the number of discrete unknowns grows exponentially with the number of populations. This means that quadrature-based methods are often not computationally efficient enough to be feasible for a large number of populations. However, the introduction of tensor train decompositions of multi-dimensional arrays in [44] opens the possibility for linearly scaling complexity laws. If the tensor train format can be successfully incorporated into numerical methods for approximating solutions to the Wright-Fisher diffusion equation it may be possible to create a method that runs in reasonable time for a larger number of populations.

1.2 Thesis Summary

In Chapters 2 and 3 of this thesis we review two mathematical ideas that are key to the work done in Chapters 4 through 6. First, in Chapter 2, we discuss the finite volume method for numerically approximating partial differential equations by partitioning the domain of the equation into finite sub-domains. Second, in chapter 3, we give a review of tensors and tensor decompositions with a specific focus on the canonical and tensor train formats.

In Chapter 4 we develop a general finite volume method for time-homogeneous convection diffusion equations with a zero source term based on the finite volume methods from [38] and [40] for convection equations. We show that our method preserves the integral and positivity of the approximation. We do not prove analytically that our method is convergent, but we do consider an example for which we give experimental evidence of convergence.

In Chapter 5 we apply the theory of tensors and the tensor train decomposition to create an algorithm that implements the finite volume method from Chapter 4 and produces the final approximation in the tensor train format. We are able to show that our method has a computational complexity that scales as $\mathcal{O}(D^4)$, where D is the number of spatial dimensions in the convection-diffusion equation. Two examples are considered in which it is found that our tensor train implementation of the finite volume method significantly outperforms matrix implementations of the same finite volume method.

Finally, in Chapter 6 we use our tensor train implementation of the finite volume method developed in Chapter 5 to approximate the solution to the Wright-Fisher diffusion equation in three populations. An in depth investigation of the impact of model parameters on computation time is also presented. We also develop an accurate and efficient method of forming tensor train approximations of certain separable functions. This method is used to efficiently compute the expected allele frequency spectrum once the solution to the Wright-Fisher diffusion equation is known in tensor train format.

Chapter 7 concludes this thesis by summarising the key results and points to areas in which more work can be done in the future.

Chapter 2

Finite Volume Methods

2.1 Conservation Equations

The finite volume method (abbreviated FVM) is a class of discretisations of various types of conservation laws. While the familiar finite difference method discretises the differential form of equations on a grid of points in the domain, the FVM discretises the integral form on a mesh of certain subsets of the domain. One of the main features of finite volume methods is that they conserve flux in a local sense. This feature means that finite volume methods are especially useful in problems where the fluxes are of importance. Such equations appear in fluid mechanics and thermodynamics [32, Chap. 2].

In this thesis we are particularly interested in a special case of the *general conservation equation*.

Definition 2.1.1 (Adapted from equation (4.1) of [67])
A function ϕ satisfies the *integral form* of the general conservation equation on a domain Ω of $\mathbb{R}^D$ if

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_K \phi(t,\boldsymbol{x})\,\mathrm{d}V(\boldsymbol{x}) + \int_{\partial K} \boldsymbol{f}(\phi,t,\boldsymbol{x})\cdot\boldsymbol{\nu}_K(\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}) = \int_K s(\phi,t,\boldsymbol{x})\,\mathrm{d}V(\boldsymbol{x}) \quad \forall K \subset \Omega, \tag{2.1}$$

where ∂K denotes the boundary surface of the set K and $\boldsymbol{\nu}_K$ is the outward pointing unit normal vector of ∂K. The domain Ω is allowed to be all of $\mathbb{R}^D$. The function $\boldsymbol{f}$ is vector valued and is called the *total flux vector* of ϕ, while s is a scalar-valued function known as the *source term*. The total flux vector measures the magnitude and direction of flow of ϕ at any point $\boldsymbol{x}$ at any time t. The source term measures the amount of value being added to or removed from ϕ at any point $\boldsymbol{x}$ at any time t. Equation (2.1) is called the *integral form* of the general conservation equation.

Remark 2.1.2
A physical interpretation of the integral form is this; the total change of ϕ in any subset K of the domain is balanced by the "flow" of ϕ value across the boundary ∂K and total amount of value injected or removed from K by the source term.

Finite volume schemes are created by discretising the integral form of the general conservation equation on a suitably chosen partition of the domain. However, in practice conservation laws are usually not given in integral form, they are given in *differential form*.

Definition 2.1.3 (Adapted from equation (4.5) of [67])
A function ϕ satisfies the *differential form* of the general conservation equation on a domain Ω of $\mathbb{R}^D$ if

$$\frac{\partial \phi}{\partial t}(t, \boldsymbol{x}) + \nabla \cdot \boldsymbol{f}(\phi, t, \boldsymbol{x}) = s(\phi, t, \boldsymbol{x}), \tag{2.2}$$

for all $t > 0$ and all $\boldsymbol{x} \in \Omega$.

A wide range of differential equations can be written as a differential form of the conservation equation. Such equations include the Navier-Stokes equations [32, Sec. 2.4] and the Wright-Fisher diffusion equation (see Proposition 6.2.1). A simpler example is given in Example 2.1.4.

Example 2.1.4
Define for $t > 0$ and $\boldsymbol{x} \in \mathbb{R}^3$ the function ϕ by

$$\frac{\partial \phi}{\partial t}(t, \boldsymbol{x}) - \frac{\partial}{\partial x_1}\left[v_1(\boldsymbol{x})\phi(t, \boldsymbol{x})\right] - \frac{\partial}{\partial x_2}\left[v_2(\boldsymbol{x})\phi(t, \boldsymbol{x})\right] - \frac{\partial}{\partial x_3}\left[v_3(\boldsymbol{x})\phi(t, \boldsymbol{x})\right] = 0 \tag{2.3}$$

for some scalar valued functions v_1, v_2 and v_3. Define the 3-dimensional vector field

$$\boldsymbol{f}(\phi, t, \boldsymbol{x}) := [-v_1(\boldsymbol{x})\phi(t, \boldsymbol{x}) \quad -v_2(\boldsymbol{x})\phi(t, \boldsymbol{x}) \quad -v_3(\boldsymbol{x})\phi(t, \boldsymbol{x})]^T.$$

Then (2.3) may be written as

$$\frac{\partial \phi}{\partial t}(t, \boldsymbol{x}) + \nabla \cdot \boldsymbol{f}(\phi, t, \boldsymbol{x}) = 0.$$

To apply finite volume methods to differential equations in differential form of (2.2), a certain equivalence of the differential and integral forms has to be established. To show the relation between the differential and integral forms the *divergence theorem* is needed.

Theorem 2.1.5 (Adapted from Section 16.9 of [54])
Let $\boldsymbol{f}$ be a D-dimensional vector field and suppose the component functions have continuous first partial derivatives. Let K be a compact region in $\mathbb{R}^D$ with piecewise continuous boundary surface ∂K. Then

$$\int_K \nabla \cdot \boldsymbol{f}(\boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) = \int_{\partial k} \boldsymbol{f}(\boldsymbol{x}) \cdot \boldsymbol{\nu}_K(\boldsymbol{x}) \, \mathrm{d}S(\boldsymbol{x}) \tag{2.4}$$

where $\boldsymbol{\nu}_K$ is the outward pointing unit normal vector of ∂K.

The divergence theorem is also known as the *Gauss-Ostragradskian theorem* (see [30, Sec. 1.1.4]). It does not hold on any arbitrary subsets of $\mathbb{R}^D$, but it holds for sets that are considered in finite volume methods. For more on the divergence theorem see [54, Sec. 15.9, Sec. 16.9]. Using the divergence theorem, the integral form of the conservation equation can be derived from the differential form.

Proposition 2.1.6
Suppose ϕ satisfies the differential form of the general conservation equation (2.2) on some open subset Ω of $\mathbb{R}^D$. If ϕ is sufficiently smooth then for any subset K of Ω, for which the divergence theorem holds, ϕ also satisfies the integral form of the general conservation equation (2.1) on K.

Proof:
The proof of Proposition 2.1.5 that is presented here is a standard proof found in most texts on the finite volume method. However, the proof is usually discussed in an expository sense. For see [49, Sec. 31.3.2] for an example.

Fix an open subset Ω of $\mathbb{R}^D$ and let K be a subset of Ω on which the divergence theorem holds. Suppose that for $t > 0$ and all $\boldsymbol{x} \in \Omega$ the function ϕ satisfies

$$\frac{\partial \phi}{\partial t}(t, \boldsymbol{x}) + \nabla \cdot \boldsymbol{f}(\phi, t, \boldsymbol{x}) = s(\phi, t, \boldsymbol{x}), \tag{2.5}$$

for some source terms s and a total flux function $\boldsymbol{f}$ with continuous first order partial derivatives. Integrating (2.5) over K yields

$$\int_K \frac{\partial \phi}{\partial t}(t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) + \int_K \nabla \cdot \boldsymbol{f}(\phi, t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) = \int_K s(\phi, t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}). \tag{2.6}$$

Consider the integral of the time derivative over K. If ϕ is integrable over K and its time derivative is also absolutely integrable over K, then by the Lebesgue dominated convergence theorem (see Theorem 12.2, Theorem 12.4 and Theorem 12.5 in [52, Chap. 12]) the time derivative can be taken outside of the integral to give

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_K \phi(t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) + \int_K \nabla \cdot \boldsymbol{f}(\phi, t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) = \int_K s(\phi, t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}). \tag{2.7}$$

This integral form is very similar to the integral form of the continuity equation, except for the fact that the flux integral is still a volume integral rather than a surface integral. By the divergence theorem the flux integral can be transformed into a surface integral and then

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_K \phi(t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) + \int_{\partial K} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_K(\boldsymbol{x}) \, \mathrm{d}S(\boldsymbol{x}) = \int_K s(\phi, t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}),$$

where $\boldsymbol{\nu}_K$ is the usual outward pointing unit normal of the boundary surface ∂K. With this the proposition is proven.

□

Remark 2.1.7
Under certain circumstances functions that satisfy the integral form of the general conservation equation also satisfy the differential form. The proof of this also depends on the Lebesgue dominated convergence theorem and the differentiability lemma (see Theorem 12.2, Theorem 12.4 and Theorem 12.5 in [52, Chap. 12]). The proof of this fact is not presented here as we are more concerned with transforming differential equations into integral equations than with the inverse.

2.2 Discretisation Process

It was mentioned in the previous section that specific finite volume methods are derived by discretising the integral form of the conservation equation on a suitably chosen partition of the domain. Therefore, the first step in the discretisation process is the selection of an appropriate partition.

Let $\mathcal{T}$ be a countable partition of subsets of the domain Ω. The set of subsets $\mathcal{T}$ is sometimes called the *mesh* and elements of the mesh are open subsets of Ω which are called *control volumes* or *cells*. The two main properties of the control volumes are that they are pairwise disjoint and fill the domain Ω. That is

$$K \cup L = \emptyset \ \ \forall\ K, L \in \mathcal{T}, \quad K \neq L, \tag{2.8}$$

and

$$\bigcup_{K \in \mathcal{T}} \mathrm{cl}(K) = \mathrm{cl}(\Omega). \tag{2.9}$$

In addition to being open it is also assumed that each control volume K is bounded, connected, polygonal and convex.

Once a partition $\mathcal{T}$ is chosen the *volume-averaged conservative form* of the conservation equation can be defined on $\mathcal{T}$.

Proposition 2.2.1 (Adapted from Section 31.3.5 of [49]**)**
Let ϕ be a function satisfying the integral form of the conservation equation (2.1) on a domain $\Omega \subset \mathbb{R}^D$. Also, let $\mathcal{T}$ be a partition if Ω satisfying (2.8) and (2.9). For each control volume $K \in \mathcal{T}$ let $\left|K\right|$ denote the D-dimensional Lebesgue measure of K and define

$$\overline{\phi}_K(t) := \frac{1}{\left|K\right|} \int_K \phi(t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}). \tag{2.10}$$

Then for all $t > 0$ the function ϕ satisfies

$$\left|K\right| \frac{\mathrm{d}\overline{\phi}_K}{\mathrm{d}t}(t) + \int_{\partial K} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_K(\boldsymbol{x}) \, \mathrm{d}S(\boldsymbol{x}) = \int_K s(\phi, t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) \ \ \forall\ K \in \mathcal{T}. \tag{2.11}$$

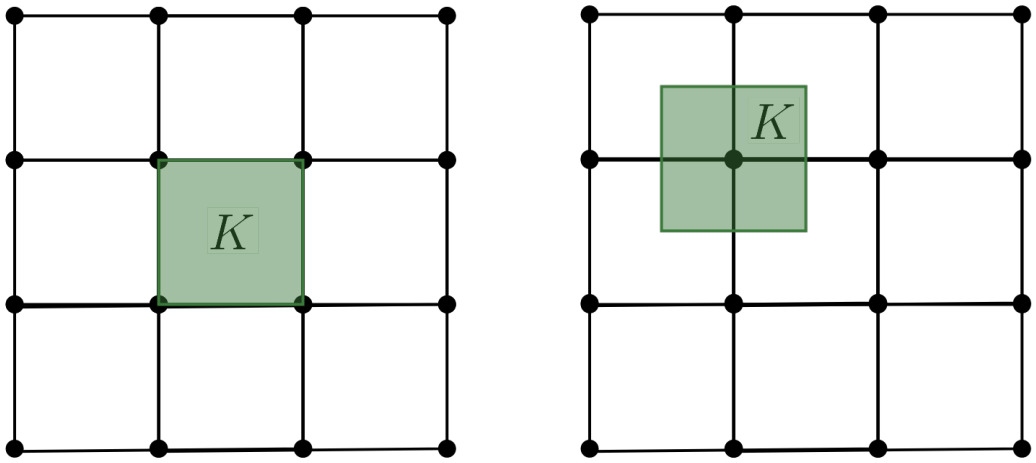
The integral equation (2.11) is called the volume-averaged conservative form of the conservation equation.

Proposition 2.2.1 follows directly from the integral form of the conservation equation and the definition of the average value (2.10).

One of the important implications of the volume-averaged conservative form is that the control volumes must remain fixed in space over time. Partitions can be structured (for example as a grid of rectangles in 2-dimensions) or unstructured (a combination of triangles and quadrilaterals in 2-dimensions). The shapes the control volumes are adapted to the geometry of the domain that is being considered and should be designed in such a way that they provide a good resolution in areas where large changes in the solution are expected to occur.

When using the volume-averaged conservative form, the volume averages $\overline{\phi}_k$ will represent the discrete unknowns that define the numerical approximation. Finite volume methods in which each discrete unknown is associated with a specific control volume or cell are known as *cell-centred* finite volume methods. These are the only methods that are considered in this thesis. There are finite volume methods in which the discrete unknowns are associated with vertices in a grid that define the mesh $\mathcal{T}$. Such finite volume methods

Figure 2.1: Difference between cell-centred and vertex-centred control volumes



(a) Cell-centred control volume. (b) Vertex-centred control volume.

are known as *cell-vertex* or *vertex-centred* methods. The difference between the types of control volumes used by these two types of methods is illustrated in Figure 2.1 .

In this thesis the domains of the differential equations that we will consider are always assumed to be rectangular. For this reason, we will also use only rectangular control volumes. There is a lot more to say about the general idea of control volumes and how to select them properly for a given problem. Specific control volumes will be discussed in subsequent examples and chapters in this thesis. However, we do not discuss general control volumes in any more detail here. For more details on control volumes see [30, Chap. 8] as well as various sections in [9].

Note that the volume-averaged conservative form is still an exact equation, no discrete approximations have been made yet. The discrete numerical method is derived from the volume-averaged conservative form by discretising each of the three terms in (2.11).

2.2.1 Time Discretisation

Consider the time derivative term in (2.11)

$$\big|K\big|\frac{\mathrm{d}\overline{\phi}_K}{\mathrm{d}t}(t). \tag{2.12}$$

This term can be discretised by taking a finite difference. Let Δt be a constant time discretisation then for all $n \in \mathbb{N}$ let $t_n := \Delta t$ and define

$$\overline{\phi}_K^n := \frac{1}{\big|K\big|}\int_K \phi(t_n, \boldsymbol{x})\,\mathrm{d}V(\boldsymbol{x}) \quad \forall\ K \in \mathcal{T}. \tag{2.13}$$

Taking a finite difference yields the approximation

$$\big|K\big|\frac{\mathrm{d}\overline{\phi}_K}{\mathrm{d}t}(t_n) \approx \big|K\big|\frac{\overline{\phi}_K^{n+1} - \overline{\phi}_K^n}{\Delta t} \quad \forall\ K \in \mathcal{T},\ n \in \mathbb{N}. \tag{2.14}$$

The variables $\overline{\phi}_K^n$ are the discrete unknowns that will be solved for to create a discrete approximation of the function ϕ. If $\overline{\phi}_K^n$ is known for each $K \in \mathcal{T}$ and all $n \in \mathbb{N}$ the piecewise constant function $\phi_{\Delta t,\mathcal{T}} : [0,\infty) \times \mathbb{R}^D \to \mathbb{R}$ defined by

$$\phi_{\Delta t,\mathcal{T}}(t,\boldsymbol{x}) := \overline{\phi}_K^n \qquad \forall (t,\boldsymbol{x}) \in [t_n, t_{n+1}) \times K, \tag{2.15}$$

gives a discrete approximation of the function ϕ. In terms of indicator functions $\phi_{\Delta t,\mathcal{T}}$ may be written as

$$\phi_{\Delta t,\mathcal{T}}(t,\boldsymbol{x}) = \sum_{n\in\mathbb{N}} 1_{T_n}(t) \sum_{K\in\mathcal{T}} \overline{\phi}_K^n 1_K(\boldsymbol{x}), \tag{2.16}$$

where 1_A denotes the indicator function for any set A (that is, 1_A is equal to 1 inside of A and zero outside) and

$$T_n = [t_n, t_{n+1}) \qquad \forall n \in \mathbb{N}. \tag{2.17}$$

More complex techniques for creating discrete approximations to ϕ using the variables $\overline{\phi}_K^n$ do exist. These techniques create approximations similar to (2.16) by using functions, such as polynomials, that are more complicated than indicator functions. In this thesis we only consider function reconstructions using indicator functions. For more interpolation techniques in finite volume methods see [49, Sec. 31.5]

Obviously, it is unreasonable to expect all $\overline{\phi}_K^n$ to be known before setting up a finite volume method. Usually only an initial value function ϕ_0 satisfying

$$\phi(0,\boldsymbol{x}) = \phi_0(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \Omega$$

is known. In this case one can define the initial condition

$$\overline{\phi}_K^0 := \frac{1}{|K|} \int_K \phi_0(\boldsymbol{x}) \,\mathrm{d}V(\boldsymbol{x}) \quad \forall\ K \in \mathcal{T}. \tag{2.18}$$

Then a recursive system of equations is derived that defines the variables $\overline{\phi}_K^n$ on all control volumes for all positive $n \in \mathbb{N}$. This system of equation is derived by discretising the other two terms in the volume-averaged conservative form. If the differential form of the conservation equation satisfied by ϕ is linear in ϕ, this system of equations will also be linear. The system of equations will be explicit if the discrete approximations of the other terms in the volume-averaged conservative form are stated in terms of $\overline{\phi}_K^n$. Similarly, the system of equations will be implicit if the discretisations are in terms of $\overline{\phi}_K^{n+1}$. This will be illustrated in 2-dimensional example in Section 2.3.

Before moving on to the discretisation of the other terms in the volume-averaged conservative form there is one fact that needs to be pointed out. The time discretisation (2.14) can be found even if the order of time derivatives and volume integrals cannot be swapped. Suppose that

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_K \phi(t,\boldsymbol{x}) \,\mathrm{d}V(\boldsymbol{x}) \neq \int_K \frac{\partial \phi}{\partial t}(t,\boldsymbol{x}) \,\mathrm{d}V(\boldsymbol{x}).$$

Approximating the derivative under the integral on the right-hand side with a finite difference yields

$$\int_K \frac{\partial \phi}{\partial t}(t_n,\boldsymbol{x}) \,\mathrm{d}V(\boldsymbol{x}) \approx \int_K \frac{\phi(t_{n+1},\boldsymbol{x}) - \phi(t_n,\boldsymbol{x})}{\Delta t} \,\mathrm{d}V(\boldsymbol{x}) = |K| \frac{\overline{\phi}_K^{n+1} - \overline{\phi}_K^n}{\Delta t} \quad \forall K \in \mathcal{T},\ n \in \mathbb{N}.$$

This means that the time discretisation of the finite volume method is still valid for functions that do not necessarily satisfy the integral form of the conservation equation.

2.2.2 Flux Terms

The next term in the volume-averaged conservative form (2.11) is the surface integral of the flux term

$$\int_{\partial K} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_K(\boldsymbol{x}) \, \mathrm{d}S(\boldsymbol{x}). \tag{2.19}$$

The first step in discretising this term is to rewrite it as the sum of integrals over the individual faces that make up the boundary ∂K. To do this, the neighbours of the control volume K have to be defined.

Definition 2.2.2 (Adapted from Section 2 of [40]**)**
Let Ω be a domain in $\mathbb{R}^D$ and let $\mathcal{T}$ be a mesh of Ω.

1. For each control volume $K \in \mathcal{T}$ define

$$\mathcal{N}_K := \{L \in \mathcal{T} | \partial K \cap \partial L \neq \emptyset \text{ and } \left|\partial K \cap \partial L\right| \neq 0\}, \tag{2.20}$$

   where $\left|\partial K \cap \partial L\right|$ denotes the $(D-1)$-dimensional Lebesgue measure of $\partial K \cap \partial L$. Each element of $\mathcal{N}_K$ is called a *neighbour* of K.

2. Let $K \in \mathcal{T}$ and define

$$E_{K,L} := \partial K \cap \partial L \quad \forall L \in \mathcal{N}_K. \tag{2.21}$$

   The set $E_{K,L}$ is called the *interface* between K and L.

Remark 2.2.3
In practice it is often assumed that the interfaces between control volumes are subsets of hyperplanes in $\mathbb{R}^D$. We assume the same in the rest of this thesis.

From the Definition 2.2.2 it follows that

$$\partial K = \bigcup_{L \in \mathcal{N}_K} E_{K,L} \quad \forall K \in \mathcal{T} \tag{2.22}$$

if we assume that ∂K does not intersect $\partial\Omega$. Then the flux integral (2.19) may be rewritten as

$$\int_{\partial K} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_K(\boldsymbol{x}) \, \mathrm{d}S(\boldsymbol{x}) = \sum_{L \in \mathcal{N}_K} \int_{E_{K,L}} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_{K,L} \, \mathrm{d}S(\boldsymbol{x}) \quad \forall\ K \in \mathcal{T}, \tag{2.23}$$

where $\boldsymbol{\nu}_{K,L}$ is the outward pointing unit normal of the boundary surface $E_{K,L}$. The normal vectors are independent of $\boldsymbol{x}$ because of the assumption made in Remark 2.2.3.

Two important properties of the boundary interfaces is that

$$E_{K,L} = E_{L,K} \quad \forall K, L \in \mathcal{T} \tag{2.24}$$

and

$$\boldsymbol{\nu}_{K,L} = -\boldsymbol{\nu}_{L,K} \quad \forall L \in \mathcal{N}_K,\ \forall K \in \mathcal{T}. \tag{2.25}$$

From these two properties it follows that for all $K \in \mathcal{T}$ and all $L \in \mathcal{N}_K$

$$\int_{E_{K,L}} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_{K,L} \, \mathrm{d}S(\boldsymbol{x}) = -\int_{E_{L,K}} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_{L,K} \, \mathrm{d}S(\boldsymbol{x}). \tag{2.26}$$

This implies that internal flux terms cancel each other out. Under certain circumstances (2.26) implies that the integral of the numerical approximation is preserved (see Lemma-4.1.8). For this reason (2.26) is a useful property that should be retained when the surface integrals in (2.23) are discretised .

Reformulating the surface integral of the flux term (2.19) as a sum of integrals over the boundary interfaces as in (2.23) is a key step in deriving any finite volume method. However, there is no one general technique for discretising the flux integrals for any arbitrary flux term. Discretisations depend on the type of mesh $\mathcal{T}$ and the specific form of the flux vector $\boldsymbol{f}$. An example discretisation for a first order differential equation is given below.

Example 2.2.4
Suppose ϕ satisfies

$$\frac{\partial \phi}{\partial t}(t, \boldsymbol{x}) + \nabla \cdot [\phi(t, \boldsymbol{x}) \boldsymbol{v}(\boldsymbol{x})] = 0 \tag{2.27}$$

for all $t > 0$ and $\boldsymbol{x} \in \Omega$ for some domain $\Omega \subset \mathbb{R}^D$. This is a D-dimensional version of the differential equation in Example 2.1.4. The total flux vector of this differential equation is

$$\boldsymbol{f}(\phi, t, \boldsymbol{x}) := \phi(t, \boldsymbol{x}) \boldsymbol{v}(\boldsymbol{x}), \quad \forall t > 0, \ \boldsymbol{x} \in \Omega.$$

Let $\mathcal{T}$ be any mesh of Ω then for every $K \in \mathcal{T}$ we have

$$\begin{aligned} \int_{\partial K} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_K(\boldsymbol{x}) \, \mathrm{d}S(\boldsymbol{x}) &= \sum_{L \in \mathcal{N}_K} \int_{E_{K,L}} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_{K,L} \, \mathrm{d}S(\boldsymbol{x}) \\ &= \sum_{L \in \mathcal{N}_K} \int_{E_{K,L}} \phi(t, \boldsymbol{x}) \left[\boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{\nu}_{K,L} \right] \mathrm{d}S(\boldsymbol{x}). \end{aligned} \tag{2.28}$$

The integral in (2.28) is still an exact expression, but for this problem it is the expression that is discretised. A common discretisation of (2.28) is the *upwinding scheme* from [38]. Define

$$v_{K,L} := \int_{E_{K,L}} \boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{\nu}_{K,L} \, \mathrm{d}S(\boldsymbol{x}) \quad \forall L \in \mathcal{N}_K, \ \forall K \in \mathcal{T} \tag{2.29}$$

then using the time discretised volume-averaged estimates $\overline{\phi}_K^n$ from (2.18) take the approximation

$$\int_{E_{K,L}} \phi(t_n, \boldsymbol{x}) \left[\boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{\nu}_{K,L} \right] \mathrm{d}S(\boldsymbol{x}) \approx \max\{0, v_{K,L}\} \overline{\phi}_K^n + \min\{0, v_{K,L}\} \overline{\phi}_L^n. \tag{2.30}$$

It follows from (2.24) and (2.25) that $v_{K,L} = -v_{L,K}$ which implies that (2.26) is maintained when approximating the flux integrals by (2.30).

The upwinding discretisation scheme is a useful scheme for discretising first order derivatives appearing in the differential form of the conservation equation because the derivative is not present in the integral form due to the divergence theorem. Second order derivatives appearing in the differential form become first order derivatives on the boundary interfaces in the integral form. Such terms can be discretised using finite differences similar to the time discretisations in Section 2.2.1. This will be explained in more detail in

Chapter 4 and a simple example is given in Section 2.3. For more details on discretisation of the flux terms see [49, Sec. 31.7], [9, Chap. 1] and [10, Sec. 4.4].

Before moving on to discretisation of the source terms in the next subsection it useful to point out how zero flux boundary conditions are imposed using the sum of surface integrals (2.23). Suppose that

$$\boldsymbol{f}(\phi, t, \boldsymbol{x}) = \boldsymbol{0} \quad \forall t > 0, \boldsymbol{x} \in \partial\Omega. \tag{2.31}$$

Also suppose that a control volume K satisfies

$$\partial K \cap \partial\Omega \neq \emptyset \quad \text{and} \quad \left|\partial K \cap \partial\Omega\right| \neq 0.$$

Let $\mathcal{E}_K$ be the set containing all boundary faces of K that are also in $\partial\Omega$. Then ∂K can be decomposed as

$$\left(\bigcup_{E\in\mathcal{E}_K} E\right) \cup \left(\bigcup_{L\in\mathcal{N}_K} E_{K,L}\right)$$

and (2.23) becomes

$$\begin{aligned}\int_{\partial K} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_K(\boldsymbol{x}) \,\mathrm{d}S(\boldsymbol{x}) = &\sum_{L\in\mathcal{N}_K} \int_{E_{K,L}} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_{K,L} \,\mathrm{d}S(\boldsymbol{x}) \\ &+ \sum_{E\in\mathcal{E}_K} \int_{E} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_E \,\mathrm{d}S(\boldsymbol{x})\end{aligned}$$

where $\boldsymbol{\nu}_E$ is the outward pointing unit normal vector of E. Since $\boldsymbol{x} \in E$ implies that $\boldsymbol{x} \in \partial\Omega$ the total flux vector vanishes on all $E \in \mathcal{E}_K$. This means that the total flux through the boundaries of K is given by

$$\int_{\partial K} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_K(\boldsymbol{x}) \,\mathrm{d}S(\boldsymbol{x}) = \sum_{L\in\mathcal{N}_K} \int_{E_{K,L}} \boldsymbol{f}(\phi, t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_{K,L} \,\mathrm{d}S(\boldsymbol{x}).$$

This looks identical to (2.23), but one should remember that flux terms relating to elements of the boundary $\partial\Omega$ have vanished. So, the flux terms only have to be discretised on internal boundary faces of the control volumes. Boundaries that coincide with the boundary of Ω can be ignored.

The fact that flux terms on $\partial\Omega$ can be ignored significantly simplifies the system of equations derived by discretising the volume-averaged conservative form. However, this is a special characteristic of zero flux boundary conditions. In general, the treatment of boundary conditions is much more complicated and depends on the boundary conditions as well as the structure of the differential equation in question. More detailed treatments of boundary conditions in finite volume methods are presented in Sections 9,10,23 and 31 of [9].

2.2.3 Source Terms

The final term in the volume-averaged conservative form that needs to be discretised is the source term

$$\int_K s(\phi, t, \boldsymbol{x}) \,\mathrm{d}V(\boldsymbol{x}). \tag{2.32}$$

In the models studied in this thesis it is always assumed that the source term is zero everywhere in the domain Ω. For this reason, we omit an in depth discussion on the treatment of source terms. Instead, we mention three common types of source terms that appear in practice and comment briefly on how they are treated.

The first type of source term that occurs commonly in practice is a constant scalar source term. That is, s is independent of ϕ and there exists $\alpha \in \mathbb{R}$ such that

$$s(t, \boldsymbol{x}) = \alpha \quad \forall t > 0,\ \boldsymbol{x} \in \Omega.$$

In the case that α is negative we call s a *sink* as it represents a constant "draining" of value from the function ϕ. Constant source terms can be discretised exactly by

$$\int_K s(t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) = \big|K\big|\alpha \quad \forall K \in \mathcal{T}.$$

where $\big|K\big|$ represents the D-dimensional Lebesgue measure of the control volume K.

Another type of source term that can appear in practice is a locally integrable function that is independent of ϕ and t. Such functions include polynomials and sinusoidal functions. In this case quadrature rules are used to evaluate the volume integral (2.32). The simplest of these quadrature rules is the midpoint rule. The midpoint rule approximates the integral by

$$\int_K s(\boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) \approx \big|K\big| s(\boldsymbol{x}_K) \quad \forall K \in \mathcal{T}$$

where $\boldsymbol{x}_K$ is the centre of the control volume K. The midpoint rule corresponds to a single node quadrature rule which makes it a very simple and computationally efficient quadrature rule. Midpoint approximations apply to any integral of continuous functions that do not vary rapidly inside of the control volume K. For this reason, it is used in this thesis to approximate many integrals such as those defining the initial condition $\overline{\phi}_K^0$ in (2.18). When the midpoint rule is not an accurate estimate of the volume integral more nodes are needed in the quadrature rule. For more on the evaluation of volume integrals in finite volume methods see [10, Sec. 4.3].

Finally, s may be represented by the divergence of a vector field $\boldsymbol{s}$ in $\mathbb{R}^D$;

$$s(\phi, t, \boldsymbol{x}) = \nabla \cdot \boldsymbol{s}(\phi, t, \boldsymbol{x}).$$

Such terms occur in the Navier-Stokes equations [32, Sec. 2.4] . For simplicity assume that $\boldsymbol{s}$ is a vector field in $\mathbb{R}^D$ that is independent of t and ϕ (this is not always the case in practice) then by the divergence theorem

$$\int_K \nabla \cdot s(\boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) = \sum_{L \in \mathcal{N}_K} \int_{E_{K,L}} \boldsymbol{s}(\boldsymbol{x}) \cdot \boldsymbol{\nu}_{K,L} \, \mathrm{d}S(\boldsymbol{x}) \quad \forall K \in \mathcal{T}. \tag{2.33}$$

In the case that $\boldsymbol{s}$ is a continuous vector field a midpoint rule can be used on the boundary $E_{K,L}$. For each $K \in \mathcal{T}$ and $L \in \mathcal{N}_K$ let $\boldsymbol{x}_{K,L}$ be the centre of the the surface $E_{K,L}$. Then by (2.33) the integral of the source term may be approximated by

$$\int_K \nabla \cdot s(\boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) \approx \sum_{L \in \mathcal{N}_K} \big|E_{K,L}\big| \boldsymbol{s}(\boldsymbol{x}_{K,L}) \cdot \boldsymbol{\nu}_{K,L} \quad \forall K \in \mathcal{T},$$

where $\left|E_{K,L}\right|$ is the $(D-1)$-dimensional Lebesgue measure of the surface $E_{K,L}$. Throughout this thesis the midpoint rule is also used to approximate surface integrals such as the flux integrals in (2.29). This is due to its simplicity and the fact that the functions for which such integrals will be needed are continuous and do not vary rapidly. In cases where the midpoint rule is inaccurate more complex quadrature rules are needed to approximate surface integrals. For more on the approximation of surface integrals in finite volume methods see [10, Sec. 4.2].

With this we end our general discussion of finite volume methods and refer to the various texts that have been cited for more details on the finite volume method.

2.3 2D Example

To summarise the full process of deriving a finite volume method for a specific problem a simple finite volume method for a 2-dimensional problem is derived here.

Let Δ denote the *Laplace operator* and consider the initial value problem

$$\begin{cases} \frac{\partial \phi}{\partial t}(t,\boldsymbol{x}) - \Delta\phi(t,\boldsymbol{x}) = 0 & \forall\ t>0,\ \boldsymbol{x}\in(0,1)^2, \\ \nabla\phi(t,\boldsymbol{x}) = \boldsymbol{0} & \forall\ t>0,\ \boldsymbol{x}\in\partial(0,1)^2, \\ \phi(0,\boldsymbol{x}) = \phi_0(\boldsymbol{x}) & \forall\ \boldsymbol{x}\in(0,1)^2. \end{cases} \tag{2.34}$$

First note that the total flux vector of the differential equation in this problem is

$$\boldsymbol{f}(\phi,t,\boldsymbol{x}) = \nabla\phi(t,\boldsymbol{x}). \tag{2.35}$$

This means that the homogeneous Neumann boundary condition is equivalent to a zero flux boundary condition for this initial value problem.

To derive a finite volume method for this problem, start by defining the control volumes. For this example and in the rest of this thesis rectangular control volumes are used. Choose two positive integers I_1 and I_2, define the set of doubles

$$I_1 \times I_2 := \big\{(i_1,i_2)\ \big|\ i_1\in\{1,\dots,I_1\} \text{ and } i_2\in\{1,\dots,I_2\}\big\}.$$

Also define the spatial step sizes

$$\Delta x_1 := I_1^{-1} \quad \text{and} \quad \Delta x_2 := I_2^{-1}.$$

The control volumes are the rectangles defined by

$$\Omega(i_1,i_2) := \big((i_1-1)\Delta x_1, i_1\Delta x_1\big) \times \big((i_2-1)\Delta x_2, i_2\Delta x_2\big) \quad \forall (i_1,i_2)\in I_1\times I_2. \tag{2.36}$$

The boundaries of these control volumes consist of sections of vertical and horizontal lines. Define the vertical line sections

$$\partial_1\Omega(i_1,i_2) := \big\{(x_1,x_2)\in(0,1)^2 \big| x_1 = i_1\Delta x_1,\ (i_2-1)\Delta x_2 \le x_2 \le i_2\Delta x_2\big\}$$

for all $i_1\in\{0,1,\dots,I_1\}$ and all $i_2\in\{1,\dots,I_2\}$. Similarly define the horizontal line sections

$$\partial_2\Omega(i_1,i_2) := \big\{(x_1,x_2)\in(0,1)^2 \big| x_2 = i_2\Delta x_2,\ (i_1-1)\Delta x_1 \le x_1 \le i_1\Delta x_1\big\}$$

for all $i_2 \in \{1, \ldots, I_1\}$ and all $i_2 \in \{0, 1, \ldots, I_2\}$. For each dimension $d = 1, 2$ the set of boundaries $\partial_d\Omega(i_1, i_2)$ the index i_d is extended to $i_d = 0$ as these represent boundaries on the $x_d = 0$ boundary. The boundaries with $i_d = I_D$ are on the $x_d = 1$ boundary. The boundaries of the control volumes can then be decomposed by

$$\partial\Omega(i_1, i_2) = \partial_1\Omega(i_1 - 1, i_2) \cup \partial_2\Omega(i_1, i_2 - 1) \cup \partial_1\Omega(i_1, i_2) \cup \partial_2\Omega(i_1, i_2) \tag{2.37}$$

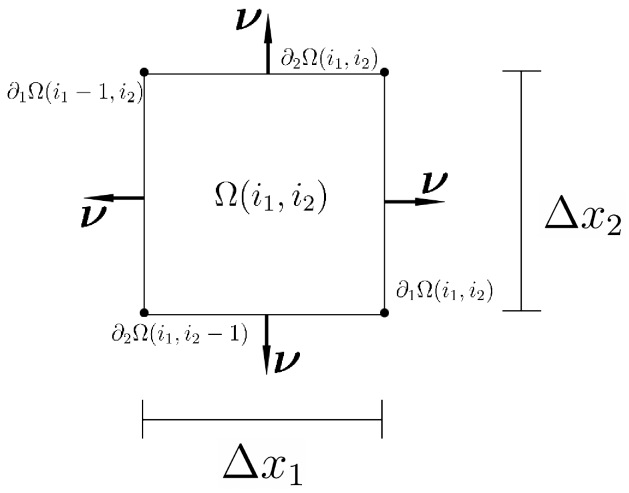
for all $(i_1, i_2) \in I_1 \times I_2$.

The outward pointing unit normal vectors of the internal boundaries are given by

$$\boldsymbol{\nu}_{(i_1,i_2)}(\boldsymbol{x}) = \begin{cases} [1\ 0]^T & \boldsymbol{x} \in \partial_1\Omega(i_1, i_2), \\ [0\ 1]^T & \boldsymbol{x} \in \partial_2\Omega(i_1, i_2), \\ [-1\ 0]^T & \boldsymbol{x} \in \partial_1\Omega(i_1 - 1, i_2), \\ [0\ -1]^T & \boldsymbol{x} \in \partial_2\Omega(i_1, i_2 - 1). \end{cases} \tag{2.38}$$

Figure 2.2 below shows a visual summary of the control volume used in this example.

Figure 2.2: A 2-dimensional rectangular control volume



Once the control volumes are defined the explicit volume-averaged conservative form of this problem can be found. Define the volume-averages

$$\overline{\phi}(t, i_1, i_2) := \left|\Omega(i_1, i_2)\right|^{-1} \int_{\Omega(i_1,i_2)} \phi(t, \boldsymbol{x})\, \mathrm{d}V(\boldsymbol{x}) \tag{2.39}$$

then the volume-averaged conservative form for this problem is

$$\left|\Omega(i_1, i_2)\right| \frac{\mathrm{d}\overline{\phi}}{\mathrm{d}t}(t, i_1, i_2) - \int_{\partial\Omega(i_1,i_2)} \nabla\phi(t, \boldsymbol{x}) \cdot \boldsymbol{\nu}_{(i_1,i_2)}(\boldsymbol{x})\, \mathrm{d}S(\boldsymbol{x}) = 0 \quad \forall\, t > 0,\ (i_1, i_2) \in I_1 \times I_2.$$

By (2.37) and (2.38) writing the surface integral of the flux term as a sum of surface integrals gives

$$\begin{aligned}\left|\Omega(i_1,i_2)\right|\frac{\mathrm{d}\overline{\phi}}{\mathrm{d}t}(t,i_1,i_2) - \int_{\partial_1\Omega(i_1,i_2)}\frac{\partial\phi}{\partial x_1}(t,\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}) + \int_{\partial_1\Omega(i_1-1,i_2)}\frac{\partial\phi}{\partial x_1}(t,\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}) \quad (2.40)\\ - \int_{\partial_2\Omega(i_1,i_2)}\frac{\partial\phi}{\partial x_2}(t,\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}) + \int_{\partial_2\Omega(i_1,i_2-1)}\frac{\partial\phi}{\partial x_2}(t,\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}) = 0.\end{aligned}$$

To discretise the time derivative, pick a constant time step $\Delta t > 0$ and take the time discretisation

$$\frac{\mathrm{d}\overline{\phi}}{\mathrm{d}t}(t_n,i_1,i_2) \approx \frac{\overline{\phi}^{n+1}(i_1,i_2) - \overline{\phi}^{n}(i_1,i_2)}{\Delta t} \quad \forall\ (i_1,i_2)\in I_1\times I_2,\ n\in\mathbb{N}. \quad (2.41)$$

To discretise the spatial derivatives in the surface integrals a midpoint approximation and centred finite difference is taken. That is

$$\frac{\partial\phi}{\partial x_1}(t_n,\boldsymbol{x}) \approx \frac{\overline{\phi}^{n}(i_1+1,i_2) - \overline{\phi}^{n}(i_1,i_2)}{\Delta x_1} \quad \forall x\in\partial_1\Omega(i_1,i_2),\ \forall(i_1,i_2)\in I_1\times I_2,\ n\in\mathbb{N}.$$

In this case the derivative at the centre of $\partial_1\Omega(i_1,i_2)$ is taken as an approximation for the average of the derivative on the entire surface. Then the derivative is approximated by a finite difference between the centre of $\Omega(i_1,i_2)$ and $\Omega(i_1+1,i_2)$. It follows that

$$\int_{\partial_1\Omega(i_1,i_2)}\frac{\partial\phi}{\partial x_1}(t,\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}) \approx \frac{\left|\partial_1\Omega(i_1,i_2)\right|}{\Delta x_1}\left(\overline{\phi}^{n}(i_1+1,i_2) - \overline{\phi}^{n}(i_1,i_2)\right). \quad (2.42)$$

From (2.40) it is clear that these discrete flux terms will be cancelled out by adjacent control volumes. This means that local flux will still be conserved. Derivatives and integrals along the x_2-axis are treated in the same way.

For internal control volumes ($2\le i_1\le I_1-1$ and $2\le i_2\le I_2-1$) the discretised volume-averaged conservative form can be written as

$$\begin{aligned}\frac{\left|\Omega(i_1,i_2)\right|}{\Delta t}\left(\overline{\phi}^{n+1}(i_1,i_2) - \overline{\phi}^{n}(i_1,i_2)\right) &- \frac{\left|\partial_1\Omega(i_1,i_2)\right|}{\Delta x_1}\left(\overline{\phi}^{n}(i_1+1,i_2) - \overline{\phi}^{n}(i_1,i_2)\right)\\ &+ \frac{\left|\partial_1\Omega(i_1-1,i_2)\right|}{\Delta x_1}\left(\overline{\phi}^{n}(i_1,i_2) - \overline{\phi}^{n}(i_1-1,i_2)\right)\\ &- \frac{\left|\partial_2\Omega(i_1,i_2)\right|}{\Delta x_2}\left(\overline{\phi}^{n}(i_1,i_2+1) - \overline{\phi}^{n}(i_1,i_2)\right)\\ &+ \frac{\left|\partial_2\Omega(i_1,i_2-1)\right|}{\Delta x_2}\left(\overline{\phi}^{n}(i_1,i_2) - \overline{\phi}^{n}(i_1,i_2-1)\right) = 0\end{aligned}$$

This gives a linear system of equations for the internal control volumes, but not control volumes sharing a boundary with $(0,1)^2$. For such control volumes the flux term from the external boundary is set to zero. For example, if $i_1 = 1$ and and $2\le i_2\le I_2-1$ then the above linear equation becomes

$$\frac{\left|\Omega(1,i_2)\right|}{\Delta t}\left(\overline{\phi}^{n+1}(1,i_2)-\overline{\phi}^{n}(1,i_2)\right)-\frac{\left|\partial_1\Omega(1,i_2)\right|}{\Delta x_1}\left(\overline{\phi}^{n}(2,i_2)-\overline{\phi}^{n}(1,i_2)\right)$$
$$-\frac{\left|\partial_2\Omega(1,i_2)\right|}{\Delta x_2}\left(\overline{\phi}^{n}(1,i_2+1)-\overline{\phi}^{n}(1,i_2)\right)$$
$$+\frac{\left|\partial_2\Omega(1,i_2-1)\right|}{\Delta x_2}\left(\overline{\phi}^{n}(1,i_2)-\overline{\phi}^{n}(1,i_2-1)\right)=0$$

The full linear system of equations can be summarised as

$$\overline{\boldsymbol{\phi}}^{n+1}=\boldsymbol{S}\overline{\boldsymbol{\phi}}^{n}\quad\forall\, n\in\mathbb{N} \tag{2.43}$$

where $\overline{\boldsymbol{\phi}}^{n}$ is a length (I_1I_2) column vector containing the averages $\overline{\phi}^{n}(i_1,i_2)$ formed by viewing $\overline{\phi}^{n}$ as a matrix and stacking columns on top of each other. The matrix $\boldsymbol{S}$ is an $(I_1I_2)\times(I_1I_2)$ matrix given by

$$\boldsymbol{S}:=\boldsymbol{I}-\Delta t(\boldsymbol{A}_1+\boldsymbol{A}_2)$$

where $\boldsymbol{I}$ is the $(I_1I_2)\times(I_1I_2)$ identity matrix. The matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ contain the flux coefficients along the x_1-axis and x_2-axis respectively. To explicitly define the matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ the multi-dimensional indices of (i_1,i_2) have to be defined in terms of an auxiliary linear index;

$$i_1(j):=j \bmod I_1\quad\text{and}\quad i_2(j):=\frac{j-i_1(j)}{I_1}+1\quad\forall\, j\in\{1,\ldots,I_1,\ldots,I_1I_2\}. \tag{2.44}$$

The matrix $\boldsymbol{A}_1$ is a sparse matrix and its non-zero entries are given by

$$\boldsymbol{A}_1(j,j):=\begin{cases}\frac{\left|\partial_1\Omega(i_1,i_2)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_1}+\frac{\left|\partial_1\Omega(i_1-1,i_2)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_1} & \text{if } 2\le i_1(j)\le I_1-1,\\ \frac{\left|\partial_1\Omega(i_1,i_2)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_1} & \text{if } i_1(j)=1,\\ \frac{\left|\partial_1\Omega(i_1-1,i_2)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_1} & \text{if } i_1(j)=I_1.\end{cases}\qquad\forall\, j\in\{1,\ldots,I_1,\ldots,I_1I_2\}$$

$$\boldsymbol{A}_1(j,j-1):=\begin{cases}-\frac{\left|\partial_1\Omega(i_1-1,i_2)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_1} & \text{if } 2\le i_1(j),\\ 0 & \text{otherwise.}\end{cases}\qquad\forall\, j\in\{2,\ldots,I_1,\ldots,I_1I_2\}$$

$$\boldsymbol{A}_1(j,j+1):=\begin{cases}-\frac{\left|\partial_1\Omega(i_1,i_2)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_1} & \text{if } i_1(j)\le I_1-1,\\ 0 & \text{otherwise.}\end{cases}\qquad\forall\, j\in\{1,\ldots,I_1,\ldots,I_1I_2-1\}$$

Similarly, the matrix $\boldsymbol{A}_2$ is a sparse matrix and its non-zero entries are given by

$$\boldsymbol{A}_2(j,j):=\begin{cases}\frac{\left|\partial_2\Omega(i_1,i_2)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_2}+\frac{\left|\partial_2\Omega(i_1,i_2-1)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_2} & \text{if } 2\le i_2(j)\le I_2-1,\\ \frac{\left|\partial_2\Omega(i_1,i_2)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_2} & \text{if } i_2(j)=1,\\ \frac{\left|\partial_2\Omega(i_1,i_2-1)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_2} & \text{if } i_2(j)=I_2.\end{cases}\qquad\forall\, j\in\{1,\ldots,I_1,\ldots,I_1I_2\}$$

$$\boldsymbol{A}_2(j, j-I_1) := \begin{cases} -\frac{\left|\partial_2\Omega(i_1,i_2-1)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_2} & \text{if } 2 \le i_2(j), \\ 0 & \text{otherwise.} \end{cases} \qquad \forall\, j \in \{I_1+1, \ldots, I_1I_2\}$$

$$\boldsymbol{A}_2(j, j+1) := \begin{cases} -\frac{\left|\partial_2\Omega(i_1,i_2)\right|}{\left|\Omega(i_1,i_2)\right|\Delta x_2} & \text{if } i_2(j) \le I_2 - 1, \\ 0 & \text{otherwise.} \end{cases} \qquad \forall\, j \in \{1, \ldots, I_1I_2 - I_1 + 1\}$$

An example should make the definition of these matrices more clear.

Example 2.3.1

Suppose that $I_1 = I_2 = 3$, then

$$\Delta x_1 = \Delta x_2 = \frac{1}{3}.$$

Since the control volumes $\Omega(i_1, i_2)$ are all the same size we know from their definition (2.36) that

$$\left|\Omega(i_1, i_2)\right| = \Delta x_1 \Delta x_2 = \frac{1}{9}. \tag{2.45}$$

Since the boundary faces $\partial_1\Omega(i_1, i_2)$ and $\partial_2\Omega(i_1, i_2)$ are all the same length we also know

$$\left|\partial_1\Omega(i_1, i_2)\right| = \Delta x_2 = \frac{1}{3} \tag{2.46}$$

and

$$\left|\partial_2\Omega(i_1, i_2)\right| = \Delta x_1 = \frac{1}{3}. \tag{2.47}$$

For all $(i_1, i_2) \in I_1 \times I_2$ it then follows that

$$\frac{\left|\partial_1\Omega(i_1, i_2)\right|}{\left|\Omega(i_1, i_2)\right|\Delta x_1} = \frac{1}{x_1^2} = 9 \quad \text{and} \quad \frac{\left|\partial_2\Omega(i_1, i_2)\right|}{\left|\Omega(i_1, i_2)\right|\Delta x_2} = \frac{1}{x_2^2} = 9.$$

Then the matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are 9×9 matrices given exactly by

$$\boldsymbol{A}_1 = \begin{bmatrix} 9 & -9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -9 & 18 & -9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -9 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & -9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -9 & 18 & -9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -9 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & -9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -9 & 18 & -9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -9 & 9 \end{bmatrix}$$

$$\boldsymbol{A}_2 = \begin{bmatrix} 9 & 0 & 0 & -9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & -9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & -9 & 0 & 0 & 0 \\ -9 & 0 & 0 & 18 & 0 & 0 & -9 & 0 & 0 \\ 0 & -9 & 0 & 0 & 18 & 0 & 0 & -9 & 0 \\ 0 & 0 & -9 & 0 & 0 & 18 & 0 & 0 & -9 \\ 0 & 0 & 0 & -9 & 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & -9 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & -9 & 0 & 0 & 9 \end{bmatrix}$$

The finite volume method presented here extends naturally to higher dimensions by adding a flux term similar to (2.42) for each dimension. For a D-dimensional problem the matrix $\boldsymbol{A}$ will be the sum of D sparse matrices defined in a similar way to $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$.

The integer functions $i_1(j)$ and $i_2(j)$ used to convert a linear index to a 2-dimensional index can be extended to arbitrary dimensions. These index operators are useful for creating the coefficient matrices of finite volume methods in higher dimensions. In Matlab linear indices can be transformed into multi-dimensional indices by using the `ind2sub` command.

The examples considered here in this section has control volumes that have a constant area, and the boundaries have constant lengths. So, the flux coefficients could be simplified to

$$\left|\Omega(i_1, i_2)\right| = \Delta x_1 \Delta x_2, \quad \left|\partial_1\Omega(i_1, i_2)\right| = \Delta x_2, \quad \text{and} \quad \left|\partial_2\Omega(i_1, i_2)\right| = \Delta x_1.$$

We left the flux coefficients in terms of the control volume indices (i_1, i_2) to illustrate the definition of the matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ when the coefficients are not equal in all control volumes, as is often the case. In Chapter 4 a finite volume method similar to the one shown here is developed where the flux coefficients do vary between control volumes.

Finally, it is necessary to point out that the finite volume method presented in this section uses a forward time discretisation leading to an explicit Euler step in (2.48). This happened because the spatial discretisations were written in terms of $\overline{\phi}^n$ rather than $\overline{\phi}^{n+1}$. Rewriting the spatial discretisations in terms of $\overline{\phi}^{n+1}$ will yield an implicit numerical scheme. The linear system of equations defining the implicit numerical scheme can be summarised by

$$\overline{\boldsymbol{\phi}}^n = \widehat{\boldsymbol{S}}\overline{\boldsymbol{\phi}}^{n+1} \quad \forall\, n \in \mathbb{Z}^+, \tag{2.48}$$

where

$$\widehat{\boldsymbol{S}} := \boldsymbol{I} + \Delta t(\boldsymbol{A}_1 + \boldsymbol{A}_2)$$

and $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are defined in exactly the same way as before.

Chapter 3

Tensors and Tensor Decompositions

In Chapter 5 we develop an algorithm for implementing a finite volume method using tensors and tensor operations. Here we give an overview of the parts of tensor theory that relates to our work.

3.1 Notation and Basic Definitions

When we refer to a *tensor* in this thesis, we simply mean a multi-dimensional array. In this section we will cover the basic definitions of tensors and some common algebraic tensor operations. Our presentation draws heavily from the review of tensors by Kolda and Bader [29].

Definition 3.1.1

1. An *Nth-order* tensor $\boldsymbol{\mathcal{X}}$ is a multi-dimensional array with entries denoted by $\boldsymbol{\mathcal{X}}(i_1, \ldots, i_N)$ where each index i_n ranges from 1 to I_n for some positive integers $I_1, \ldots, I_N$. Each index i_n is referred to as a *mode* and I_n is the *length* or *size* of the mode. We collect the mode subscripts in a set $\mathcal{N} := \{1, \ldots, N\}$ and the size of $\boldsymbol{\mathcal{X}}$ is defined as the N-tuple $I_{\mathcal{N}} := (I_1, \ldots, I_N)$.

2. For any positive integers $N, I_1, \ldots, I_N$ the set $\mathbb{R}^{I_1 \times \cdots \times I_N}$ is defined to be the space of all Nth-order tensors with mode lengths given by $I_1, \ldots, I_N$.

3. For a given Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ a *sub-tensor* is formed by fixing a subset of indices.

4. A first-order sub-tensor is called a *fibre*, and a *mode n fibre* is a sub-tensor with all indices except the nth being fixed.

5. A Second-order sub-tensor is called a *slice*.

When working with scalars, vectors, matrices and tensors of higher orders it is useful to adopt a notation that distinguishes between these objects. So, we use normal letters (lower and upper case) for scalars e.g., N or n. For vectors (first-order tensors) we use boldface lower case letters e.g., $\boldsymbol{a}$. Matrices (second-order tensors) are denoted by upper case boldface letters e.g., $\boldsymbol{A}$. Finally, for tensors of order three or higher we use boldface Euler scripts e.g., $\boldsymbol{\mathcal{X}}$.

Examples are the best illustrations of the ideas introduced in Definition 3.1.1 and they provide an opportunity to introduce some common notational conventions.

Example 3.1.2
Let $\boldsymbol{X}$ be a 2 by 2 matrix given by

$$\boldsymbol{X} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix},$$

then $\boldsymbol{X}$ is a second-order tensor with mode lengths $I_1 = I_2 = 2$. The columns are given by the fibres

$$\boldsymbol{X}(:,1) = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \text{ and } \boldsymbol{X}(:,2) = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

We adopt the notation from Matlab [36] where colons are used to indicate that an index is unfixed. Rows are given by the fibres

$$\boldsymbol{X}(1,:) = \begin{bmatrix} 1 & 3 \end{bmatrix} \text{ and } \boldsymbol{X}(2,:) = \begin{bmatrix} 2 & 4 \end{bmatrix}.$$

Explicitly defining higher order tensors is usually done by giving the two-dimensional slices that make up the tensor. For example, we can define a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{3\times3\times3}$ by

$$\boldsymbol{\mathcal{X}}(:,:,1) = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}, \quad \boldsymbol{\mathcal{X}}(:,:,2) = \begin{bmatrix} 10 & 13 & 16 \\ 11 & 14 & 17 \\ 12 & 15 & 18 \end{bmatrix} \text{ and } \boldsymbol{\mathcal{X}}(:,:,3) = \begin{bmatrix} 19 & 22 & 25 \\ 20 & 23 & 26 \\ 21 & 24 & 27 \end{bmatrix}.$$

Placing these three matrices behind each other as illustrated in Figure-3.1 forms the full tensor $\boldsymbol{\mathcal{X}}$.

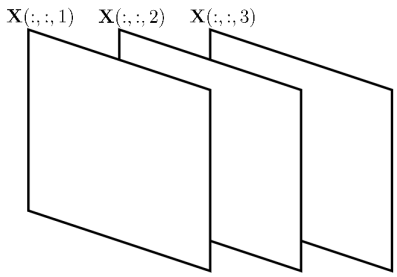


Figure 3.1: Second-order slices of a third-order tensor

Tensor addition and scalar multiplication are defined in the exact same pointwise sense as matrices. With these two operations any space of Nth-order tensors $\mathbb{R}^{I_1\times\cdots\times I_N}$ can be viewed as a vector space, in fact it can even be turned into an inner product space. The usual vector inner product and Euclidean/Frobenius norm extend nicely to tensors.

Definition 3.1.3
Let $N, I_1, \ldots, I_N$ be positive integers.

1. An inner product $\langle\cdot|\cdot\rangle$ on $\mathbb{R}^{I_1\times\cdots\times I_N}$ is defined by

$$\langle \boldsymbol{\mathcal{X}} | \boldsymbol{\mathcal{Y}} \rangle := \sum_{i_1,\ldots,i_N=1}^{I_1,\ldots,I_N} \boldsymbol{\mathcal{X}}(i_1,\ldots,i_N)\boldsymbol{\mathcal{Y}}(i_1,\ldots,i_N) \text{ for all } \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1\times\cdots\times I_N}.$$

2. The Frobenius norm on $\mathbb{R}^{I_1\times\cdots\times I_N}$ denoted by $||\cdot||_F$ is defined by

$$||\boldsymbol{\mathcal{X}}||_F := \langle \boldsymbol{\mathcal{X}} | \boldsymbol{\mathcal{X}} \rangle^{1/2}.$$

Remark 3.1.4
When working with vectors (or first-order tensors) the Frobenius norm coincides with the Euclidean norm, which is denoted $||\cdot||_2$. However for $M \times N$ matrices (or second-order tensors) it is standard practice for $||\cdot||_2$ to denote the operator norm induced by the Euclidean norm on $\mathbb{R}^N$. So even though the Frobenius norm is defined in the same way as the Euclidean norm, to avoid confusion with norms for linear operators we denote the Frobenius norm by $||\cdot||_F$.

The fact that inner products and norms can be so easily extended from matrices to tensors suggests that the space of tensors and the space of matrices are isomorphic. They are indeed isomorphic and an isomorphism between a space of tensors and a space of matrices is known as a *matricisation* or *flattening*.

Definition 3.1.5 (Adapted from Section 3.4 of [28]**)**

1. Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, let $\mathcal{R} = \{r_1, \cdots, r_L\}$ and $\mathcal{C} = \{c_1, \cdots, c_M\}$ partition the set of modes $\mathcal{N} = \{1, \cdots, N\}$. The matricisation (or flattening) of $\boldsymbol{\mathcal{X}}$ induced by the partition $\mathcal{R} \cup \mathcal{C} = \mathcal{N}$ is given by the matrix

$$\boldsymbol{X}_{(\mathcal{R}\times\mathcal{C}:I_{\mathcal{N}})} \in \mathbb{R}^{J\times K} \quad \text{with} \quad J = \prod_{n\in\mathcal{R}} I_n \quad \text{and} \quad K = \prod_{n\in\mathcal{C}} I_n.$$

   The entries of this matrix are defined by

$$(\boldsymbol{X}_{(\mathcal{R}\times\mathcal{C}:I_{\mathcal{N}})})_{jk} = x_{i_1 i_2 \cdots i_N},$$

   where

$$j = 1 + \sum_{l=1}^{L} \left[(i_{r_l} - 1) \prod_{l'=1}^{l-1} I_{r_{l'}} \right] \quad \text{and} \quad k = 1 + \sum_{m=1}^{M} \left[(i_{r_m} - 1) \prod_{m'=1}^{m-1} I_{r_{m'}} \right].$$

2. An *n-mode unfolding* of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the matricisation induced by the partitioning of modes with $\mathcal{R} = \{n\}$ and $\mathcal{C} = \mathcal{N} \setminus \mathcal{R}$. In this case the mode n fibres become the columns of the matricisation. We denote the n-mode unfolding of $\boldsymbol{\mathcal{X}}$ by $\boldsymbol{X}_n$.

The formal definition of matricisations can be clunky and quite unintuitive. Examples illustrate the idea much better.

Example 3.1.6
Consider again the third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{3\times3\times3}$ from Example 3.1.2 defined by its slices;

$$\boldsymbol{\mathcal{X}}(:,:,1) = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}, \quad \boldsymbol{\mathcal{X}}(:,:,2) = \begin{bmatrix} 10 & 13 & 16 \\ 11 & 14 & 17 \\ 12 & 15 & 18 \end{bmatrix} \quad \text{and} \quad \boldsymbol{\mathcal{X}}(:,:,3) = \begin{bmatrix} 19 & 22 & 25 \\ 20 & 23 & 26 \\ 21 & 24 & 27 \end{bmatrix}.$$

There are three n-mode unfoldings of $\boldsymbol{\mathcal{X}}$;

$$\boldsymbol{X}_1 = [\boldsymbol{\mathcal{X}}(:,:,1) \quad \boldsymbol{\mathcal{X}}(:,:,2) \quad \boldsymbol{\mathcal{X}}(:,:,3)\ ] = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 & 25 \\ 2 & 5 & 8 & 11 & 14 & 17 & 20 & 23 & 26 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 & 27 \end{bmatrix},$$

$$\boldsymbol{X}_2 = [\boldsymbol{\mathcal{X}}(:,:,1)^T \ \boldsymbol{\mathcal{X}}(:,:,2)^T \ \boldsymbol{\mathcal{X}}(:,:,3)^T] = \begin{bmatrix} 1 & 2 & 3 & 10 & 11 & 12 & 19 & 20 & 21 \\ 4 & 5 & 6 & 13 & 14 & 15 & 22 & 23 & 24 \\ 7 & 8 & 9 & 16 & 17 & 18 & 25 & 26 & 27 \end{bmatrix},$$

$$\boldsymbol{X}_3 = \begin{bmatrix} \text{vec}(\boldsymbol{\mathcal{X}}(:,:,1))^T \\ \text{vec}(\boldsymbol{\mathcal{X}}(:,:,2))^T \\ \text{vec}(\boldsymbol{\mathcal{X}}(:,:,3))^T \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 \end{bmatrix}.$$

We used $\text{vec}(\cdot)$ to denote the usual vectorisation of a matrix formed by stacking columns on top of each other to form a column vector. For third order tensors the other matricisations are simply $\text{vec}(X_1)$, $\text{vec}(X_1)^T$,and transposes of $\boldsymbol{X}_1, \boldsymbol{X}_2$ and $\boldsymbol{X}_3$. As the order increases the number of matricisations increases exponentially. However, the most important matricisations are the n-mode unfoldings.

Matrix-matrix and matrix-vector multiplication are fundamental to matrix algebra. These operations are extended to tensors by an operation called the *n-mode product*.

Definition 3.1.7 (Adapted from Section 2.5 of [29].)
Let $\boldsymbol{\mathcal{X}}$ be an Nth-order tensor in $\mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\boldsymbol{A}$ a $J \times I_n$ matrix for some positive integer J and $n \in \{1, \ldots, N\}$. The n-mode product of $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{M}$ denoted by $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{M}$ has size $(I_1, \ldots, I_{n-1}, J, I_{n+1}, \ldots, I_N)$ and its entries are given by

$$[\boldsymbol{\mathcal{X}} \times_n \boldsymbol{M}](i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, I_N) = \sum_{i_n=1}^{I_N} \boldsymbol{\mathcal{X}}(i_1, \ldots, i_n, \ldots, i_N)\boldsymbol{M}(j, i_n). \tag{3.1}$$

There are a few important results that follow directly from (3.1).

Corollary 3.1.8 (Adapted from [28])

1. *For a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a matrix $\boldsymbol{M} \in \mathbb{R}^{J \times I_n}$, computing the n-mode product requires $\mathcal{O}(JI^N)$ floating point operations. Here I denotes the maximum of all $I_1, \ldots, I_N$.*

2. *The n-mode product is commutative. For any Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ if $\boldsymbol{A} \in \mathbb{R}^{J_n \times I_n}$ and $\boldsymbol{B} \in \mathbb{R}^{J_m \times I_m}$ with $n \neq m$ and $1 \leq n, m \leq N$ we have*

$$(\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A}) \times_m \boldsymbol{B} = (\boldsymbol{\mathcal{X}} \times_m \boldsymbol{B}) \times_n \boldsymbol{A}.$$

3. *For any Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ if $\boldsymbol{A} \in \mathbb{R}^{J \times I_n}$ and $\boldsymbol{B} \in \mathbb{R}^{K \times J}$ we have*

$$\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A} \times_n \boldsymbol{B} = \boldsymbol{\mathcal{X}} \times_n (\boldsymbol{BA}).$$

4. *The n-mode product extends matrix multiplication. If $\boldsymbol{A} \in \mathbb{R}^{I \times J_1}$ and $\boldsymbol{B} \in \mathbb{R}^{I_2 \times I}$ then*

$$\boldsymbol{A} \times_1 \boldsymbol{B} = \boldsymbol{BA},$$

and if $\boldsymbol{A} \in \mathbb{R}^{I_1 \times J}$ and $\boldsymbol{B} \in \mathbb{R}^{J \times J_2}$ then

$$\boldsymbol{A} \times_2 \boldsymbol{B} = \boldsymbol{AB}.$$

There are a few more matrix and tensor operations that will be useful to us in the discussions that follow later.

Definition 3.1.9 (Adapted from [29])

1. For any two matrices $\boldsymbol{A} \in \mathbb{R}^{I_1 \times J_1}$ and $\boldsymbol{B} \in \mathbb{R}^{I_2 \times J_2}$ their *Kronecker product* is denoted by $\boldsymbol{A} \otimes \boldsymbol{B}$ which is an $(I_1 I_2) \times (J_1 J_2)$ matrix given by

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}(1,1)\boldsymbol{B} & \boldsymbol{A}(1,2)\boldsymbol{B} & \cdots & \boldsymbol{A}(1,J_1)\boldsymbol{B} \\ \boldsymbol{A}(2,1)\boldsymbol{B} & \boldsymbol{A}(2,2)\boldsymbol{B} & \cdots & \boldsymbol{A}(2,J_1)\boldsymbol{B} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}(I_1,1)\boldsymbol{B} & \boldsymbol{A}(I_1,2)\boldsymbol{B} & \cdots & \boldsymbol{A}(I_1,J_1)\boldsymbol{B} \end{bmatrix}. \tag{3.2}$$

2. For any two matrices $\boldsymbol{A} \in \mathbb{R}^{I_1 \times K}$ and $\boldsymbol{B} \in \mathbb{R}^{I_2 \times K}$ their *Khatri-Rao product* is denoted by $\boldsymbol{A} \odot \boldsymbol{B}$ which is an $(I_1 I_2) \times K$ matrix given by

$$\boldsymbol{A} \odot \boldsymbol{B} = \left[\boldsymbol{a}_1 \otimes \boldsymbol{b}_1 \ \cdots \ \boldsymbol{a}_K \otimes \boldsymbol{b}_K \right],$$

   where $\boldsymbol{a}_k$ and $\boldsymbol{b}_k$ are th kth colums of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively.

3. For any two tensors $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ their *Hadamard product* is denoted by $\boldsymbol{\mathcal{X}} * \boldsymbol{\mathcal{Y}}$ which is an Nth-order tensor with size $(I_1, \ldots, I_N)$ and entries given by

$$[\boldsymbol{\mathcal{X}} * \boldsymbol{\mathcal{Y}}](i_1, \ldots, i_N) = \boldsymbol{\mathcal{X}}(i_1, \ldots, I_N)\boldsymbol{\mathcal{Y}}(i_1, \ldots, I_N).$$

4. For any two vectors $\boldsymbol{a}_1 \in R^{I_1}$ and $\boldsymbol{a}_2 \in \mathbb{R}^{I_2}$ their *outer product* is denoted by $\boldsymbol{a}_1 \circ \boldsymbol{a}_2$ which is an $I_1 \times I_2$ matrix with entries given by

$$[\boldsymbol{a}_1 \circ \boldsymbol{a}_2](i_1, i_2) = \boldsymbol{a}_1(i_1)\boldsymbol{a}_2(i_2).$$

   This can be extend to N vectors $\boldsymbol{a}_n \in \mathbb{R}^{I_n}$ to form an Nth-order tensor given by

$$[\boldsymbol{a}_1 \circ \ldots \circ \boldsymbol{a}_N](i_1, \ldots, i_N) = \boldsymbol{a}_1(i_1) \ldots \boldsymbol{a}_N(i_N).$$

Outer and Hadamard products will be used extensively later when we implement tensors in an algorithm (Algorithm 5) for approximating the solution of a particular class of partial differential equations. We close this section by stating an important property of the Kronecker product that will be rather useful for reducing the complexity of Hadamard products.

Corollary 3.1.10 (Adapted from [63])
For any matrices $\boldsymbol{A} \in \mathbb{R}^{I \times J}, \boldsymbol{B} \in \mathbb{R}^{L \times K}, \boldsymbol{C} \in \mathbb{R}^{J \times M}$ *and* $\boldsymbol{D} \in \mathbb{R}^{K \times N}$ *then we have that*

$$(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = (\boldsymbol{AC}) \otimes (\boldsymbol{BD}).$$

The resultant tensor has size $(I_1, \ldots, I_N)$.

3.2 Tensor Decompositions

In numerical computations we often encounter the so-called *curse of dimensionality*. The curse of dimensionality is a phenomenon where the number of unknowns that we need to compute grows exponentially with the dimension of the problem. An Nth-order tensor with maximum mode size I has $\mathcal{O}(I^N)$ entries. This means that tensor operations require at least $\mathcal{O}(I^N)$ floating point operations to compute. Matrices also encounter this

curse. For example, when using standard matrix methods to implement the finite volume method the number of control volumes grows exponentially with the dimension of the PDE. So, the size of the matrices used in the implementation will grow exponentially with the dimension of the PDE. To an extent, we can remedy the curse of dimensionality in matrices using decompositions such as the Singular value decomposition (SVD), the QR-decomposition, LU-factorisation, Skeleton Decomposition (see [13]) and Cholesky factorisation, (see Sections 3.4, 5.3 and 7.4 of [48] and various lectures in [57]).

3.2.1 Canonical Decomposition

Matrix decompositions are methods of breaking down matrices into a few components that allow us to fully reconstruct the original matrix. All of these decompositions try to approximate a given matrix by a product of smaller matrices. For example the reduced singular value decomposition (see Lecture 4 in [57]) of a real matrix $\boldsymbol{M} \in \mathbb{R}^{I\times J}$ with R linearly independent rows and columns (so a rank-R matrix) is given by a matrix $\boldsymbol{U} \in \mathbb{R}^{I\times R}$ with orthonormal columns, a rectangular diagonal matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{R\times R}$ with non-increasing positive real entries on the diagonal, and another matrix $\boldsymbol{V} \in \mathbb{R}^{R\times J}$ with orthonormal rows. Then $\boldsymbol{U}, \boldsymbol{\Sigma}$ and $\boldsymbol{V}$ satisfy

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}.$$

Denoting the columns of $\boldsymbol{U}$ by $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_R$, the rows of $\boldsymbol{V}$ by $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_R$ and the diagonal of $\boldsymbol{\Sigma}$ by $\boldsymbol{\sigma}$ allows us to write

$$\boldsymbol{M} = \sum_{r=1}^{R} \boldsymbol{\sigma}(r)\boldsymbol{u}_r \circ \boldsymbol{v}_r. \tag{3.3}$$

The entries of $\boldsymbol{\Sigma}$ are called the *singular values* of $\boldsymbol{M}$. An effective approximation of $\boldsymbol{M}$ can then be obtained by dropping terms in (3.3) with small singular values $\boldsymbol{\sigma}(r)$.

The fact that outer products can be extended to more than two vectors to form a tensor allows for the definition of a similar notion of rank for tensors.

Definition 3.2.1 (Adapted from [29]**)**
The *canonical rank* of an Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times\ldots\times I_N}$ is the smallest positive integer R for which there exists a matrix $\boldsymbol{A}_n \in \mathbb{R}^{I_n\times R}$ with unit vector columns denoted by $\boldsymbol{a}_n^1, \ldots, \boldsymbol{a}_n^R$ for each $n \in \{1, \ldots, N\}$ and a vector $\boldsymbol{\lambda} \in \mathbb{R}^R$ such that

$$\boldsymbol{\mathcal{X}} = \sum_{r=1}^{R} \boldsymbol{\lambda}(r)\boldsymbol{a}_1^r \circ \ldots \circ \boldsymbol{a}_N^r. \tag{3.4}$$

The matrices $\boldsymbol{A}_n$ are called the *factor matrices* and the entries of the vector $\boldsymbol{\lambda}$ the *weights*. It is common practice to arrange the factor matrices and the weights so that

$$\boldsymbol{\lambda}(1) \geq \ldots \geq \boldsymbol{\lambda}(R).$$

We denote the fact that (3.4) holds for the factor matrices by

$$\boldsymbol{\mathcal{X}} = [\![\boldsymbol{\lambda}; \boldsymbol{A}_1, \ldots, \boldsymbol{A}_N]\!]. \tag{3.5}$$

If the factor matrices $\boldsymbol{A}_n$ and the weights $\boldsymbol{\lambda}$ are known we say that we have the tensor $\boldsymbol{\mathcal{X}}$ in the *canonical format*.

The reduced singular value decomposition provides a canonical decomposition for matrices. Also note that (3.3) shows that the canonical rank for a matrix is equivalent to the usual rank of the matrix given by the number of linearly independent columns (or rows).

Many authors refer to the canonical decomposition as the *parallel factor* decomposition, [19]. For this reason the canonical decomposition has come to be known as the *CANDECOMP/PARAFAC* mode or format. In shortened form it is known as the *CP format*. The main benefit of the canonical format is that it can greatly reduce the number of values needed to completely reconstruct the full tensor. For an Nth-order tensor with maximum mode length I and canonical rank R in canonical format only $\mathcal{O}(NIR)$ values need to be stored, as opposed to the $\mathcal{O}(I^N)$ entries in the full tensor.

While the canonical format has the ability to greatly compress the storage needed for a tensor, it also has a few detrimental properties. First of all, canonical decompositions are not unique. It was shown in [8] by Eckart and Young that for any rank-R matrix $\boldsymbol{X}$ and positive integer $k < R$, the optimal rank-k approximation of $\boldsymbol{X}$ is found by taking the first k terms in (3.3). It was shown in [27] that this result does not extend to higher order tensors. This was done by providing a third-order tensor with an optimal rank-one approximation that does not appear in the optimal rank-two approximation. By an optimal rank-k approximation of a tensor we mean the closest tensor with canonical rank k.

Another problem with the canonical rank is that the best rank-k approximation might not even exist. In fact, some tensors may even be approximated arbitrarily closely by a tensor of lower canonical rank. This is shown by an example from [47]. Consider a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ with canonical rank three which is defined by

$$\boldsymbol{\mathcal{X}} = \boldsymbol{a}_1 \circ \boldsymbol{b}_1 \circ \boldsymbol{c}_2 + \boldsymbol{a}_1 \circ \boldsymbol{b}_2 \circ \boldsymbol{c}_1 + \boldsymbol{a}_2 \circ \boldsymbol{b}_1 \circ \boldsymbol{c}_1$$

where $\boldsymbol{a}_1, \boldsymbol{a}_2 \in \mathbb{R}^{I_1}$, $\boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathbb{R}^{I_2}$ and $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathbb{R}^{I_3}$. Consider a class of tensors of the form;

$$\boldsymbol{\mathcal{Y}}(\alpha) = \alpha \left(\boldsymbol{a}_1 + \frac{1}{\alpha} \boldsymbol{a}_2 \right) \circ \left(\boldsymbol{b}_1 + \frac{1}{\alpha} \boldsymbol{b}_2 \right) \circ \left(\boldsymbol{c}_1 + \frac{1}{\alpha} \boldsymbol{c}_2 \right) - \alpha \boldsymbol{a}_1 \circ \boldsymbol{b}_1 \circ \boldsymbol{c}_1,$$

for $\alpha \geq 0$. These tensors have canonical rank two. Moreover, by expanding the brackets in the definition of $\boldsymbol{\mathcal{Y}}(\alpha)$ it can be shown that

$$||\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}(\alpha)||_F = \frac{1}{\alpha} \left|\left| \boldsymbol{a}_2 \circ \boldsymbol{b}_2 \circ \boldsymbol{c}_1 + \boldsymbol{a}_2 \circ \boldsymbol{b}_1 \circ \boldsymbol{c}_2 + \boldsymbol{a}_1 \circ \boldsymbol{b}_2 \circ \boldsymbol{c}_2 + \frac{1}{\alpha} \boldsymbol{a}_2 \circ \boldsymbol{b}_2 \circ \boldsymbol{c}_2 \right|\right|_F .$$

Since the vectors $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are independent of α the difference $||\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}(\alpha)||_F$ can be made arbitrarily small, by making α arbitrarily large.

Definition 3.2.2
An Nth-order tensor $\boldsymbol{\mathcal{X}}$ with canonical rank R is called *degenerate* if it can be approximated arbitrarily closely by a tensor with a canonical rank less than R.

The existence of degenerate tensors has important implications for algorithms that attempt to calculate a canonical decomposition of a tensor. Firstly, it was shown in [21] that computing the canonical rank for an arbitrary tensor is an NP-hard problem. Therefore, it might not be possible to solve for the canonical rank of an arbitrary tensor in finite time. Kolda and Bader note in [29] that even some of the most widely used algorithms

that attempt to compute a canonical decomposition are not guaranteed to work. They often get stuck at local minima. It can also be shown that there might not even exist a best rank-k approximation for tensors of order three or higher, [5]. Despite these drawbacks, the canonical decomposition has seen wide spread use in analytic chemistry ([39],[41],[12]). With these remarks we end our overview of the canonical decomposition and refer to [29] for more on this topic, including more detailed results on the canonical decomposition and applications.

3.2.2 Tensor Trains

The canonical decomposition is the simplest and most natural extension of the idea of the singular value matrix decomposition. However, due to technical drawbacks many attempts have been made to find more stable, if perhaps more complex, decompositions. Decomposition that require $\mathcal{O}(N)$ values to exactly recreate an Nth-order tensor are clearly desirable. One such decomposition, introduced in [44] by Oseledets, is the *Tensor Train* (TT) decomposition or more commonly *TT format*. The TT format is similar to the singular value decomposition in that it uses products of matrices to approximate the full tensor. Oseledets provides many new algorithms and procedures for performing algebraic operations in TT format in the [44], for which we provide a brief review here.

Definition 3.2.3 (Adapted from [44]**)**
Let $\boldsymbol{\mathcal{X}}$ be an Nth-order tensor in $\mathbb{R}^{I_1 \times \ldots \times I_N}$. A *tensor train decomposition* of $\boldsymbol{\mathcal{X}}$ is a set of third-order tensors $\boldsymbol{\mathcal{U}}_n$ in $\mathbb{R}^{R_{n-1} \times I_n \times R_n}$, for $n \in \{1, \ldots, N\}$, such that $R_0 = R_N = 1$ and

$$\boldsymbol{\mathcal{X}}(i_1, \ldots, i_N) = \boldsymbol{\mathcal{U}}_1(:, i_1, :)\boldsymbol{\mathcal{U}}_2(:, i_2, :) \ldots \boldsymbol{\mathcal{U}}_N(:, i_N, :), \tag{3.6}$$

for each $i_n \in \{1, \ldots, I_n\}$, for all $n \in \{1, \ldots, N\}$. In index form 3.6 is given by

$$\boldsymbol{\mathcal{X}}(i_1, \ldots, i_N) = \sum_{r_0, \ldots, r_N = 1}^{R_0, \ldots, R_N} \boldsymbol{\mathcal{U}}_1(r_0, i_1, r_1)\boldsymbol{\mathcal{U}}_2(r_1, i_2, r_2) \ldots \boldsymbol{\mathcal{U}}_N(r_{N-1}, i_N, r_N). \tag{3.7}$$

The mode lengths R_n are called the *TT ranks* of the decomposition. The third-order tensors $\boldsymbol{\mathcal{U}}_n$ are called the *cores* of the decomposition. We usually abbreviate "tensor train decomposition" to *TT decomposition*.

From the definition of the TT decomposition it is clear that storing a tensor in TT format requires the storage of only $\mathcal{O}(NIR^2)$ values, where I is the maximum mode length and R is the maximum TT rank. This is still linear in dimension, just as the canonical decomposition requires storage of $\mathcal{O}(NIR)$ values for a tensor with canonical rank R. So, the storage required for the TT format is comparable to the storage required for the canonical format. However, we will see below that the TT format has many desirable properties that the canonical format does not have. But first it should be noted that having a tensor in the canonical format automatically means we have that same tensor in TT format. This is shown by Theorem 3.2.4 below.

Theorem 3.2.4 (Adapted from Section 3.1 in [44])
Suppose that $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ *has canonical rank-R and canonical decomposition*

$$\boldsymbol{\mathcal{X}} = [\![\boldsymbol{\lambda}; \boldsymbol{A}_1, \ldots, \boldsymbol{A}_N]\!],$$

for some $\lambda \in \mathbb{R}^R$ *and factor matrices* $\boldsymbol{A}_n \in \mathbb{R}^{I_n \times R}$*, for each* $n = 1, \ldots, N$*. Then* $\boldsymbol{\mathcal{X}}$ *has a TT decomposition with all TT ranks equal to R with internal cores given by*

$$\boldsymbol{\mathcal{X}}_n(:, i_n, :) = \mathrm{diag}(\boldsymbol{A}_n(i_n, :)), \quad n = 2, \ldots, N-1$$

and boundary cores given by

$$\boldsymbol{\mathcal{X}}_1(i_1, :) = \boldsymbol{A}_1(i_1, :)\mathrm{diag}(\boldsymbol{\lambda}), \quad \textit{and} \quad \boldsymbol{\mathcal{X}}_N(:, i_N) = \boldsymbol{A}_N(i_N, :)^T.$$

An important corollary of Theorem 3.2.4 is that certain functions can be approximated by a discrete tensor of low rank.

Proposition 3.2.5
Suppose f *is a scalar valued function on* $\mathbb{R}^D$ *that is also separable, i.e. there exists a function* $f_d : \mathbb{R} \to \mathbb{R}$ *for each* $d = 1, \ldots, D$ *such that*

$$f(\boldsymbol{x}) = \prod_{d=1}^{D} f_d(x_d) \quad \forall\ \boldsymbol{x} \in \mathbb{R}. \tag{3.8}$$

For each $d = 1, \ldots, D$ *let* I_d *be a positive integer and let* $X_d := \{x_d^{(i_d)}\}_{i_d=1}^{I_d}$ *be a strictly increasing sequence of real numbers. Then the tensor*

$$\boldsymbol{\mathcal{F}}(i_1, \ldots, i_D) := f(x_1^{(i_1)}, \ldots, x_D^{(i_D)}), \tag{3.9}$$

has an exact TT decomposition with all TT ranks equal to 1. Moreover, the cores of that decomposition are the vectors

$$\boldsymbol{f}_d(i_d) = f_d(x_d^{(i_d)}), \quad \forall i_d \in \{1, \ldots, I_d\},\ \forall d \in \{1, \ldots, D\}.$$

Proposition 3.2.5 follows directly by computation from (3.8), (3.9) and Theorem 3.2.4.

Remark 3.2.6
Proposition 3.2.5 allows us to easily form approximate TT decompositions of functions if we can write them as the sum or product of separable functions. When we refer to the TT ranks of a function, we mean the induced TT ranks of a TT approximation of the function on a specific discrete grid.

Remark 3.2.7
In the discussion that follows we will be dealing with a certain type of unfolding quite often. So, to tidy up notation we define some new notation. Let $\boldsymbol{\mathcal{X}}$ be an Nth-order tensor with size $I_{\mathcal{N}} = (I_1, \ldots, I_N)$. For any $n \in \{1, \ldots, N\}$ set $\mathcal{R} = \{1, \ldots, n\}$ and $\mathcal{C} = \{n+1, \ldots, N\}$ then $\boldsymbol{X}_{\mathcal{R} \times \mathcal{C}; I_{\mathcal{N}}}$ (as defined by Definition 3.1.5.1) is denoted by $\boldsymbol{X}^{(n)}$. Entries are denoted by $\boldsymbol{\mathcal{X}}(i_1, \ldots, i_n | i_{n+1}, \ldots, i_N)$.

It is shown in [44] that every tensor does in fact have a TT decomposition. This proof is not only constructive but also requires a finite number of steps. We sketch the proof here and present the general construction in Algorithm 1.

Theorem 3.2.8 (Theorem 2.1 in [44])
Let $\boldsymbol{\mathcal{X}}$ is an Nth-order tensor and suppose that $\text{rank}(\boldsymbol{X}^{(n)}) = R_n$ *for each* $n = 1, \ldots, N$, *then there exists a TT decomposition of $\boldsymbol{\mathcal{X}}$ with TT ranks bounded above by R_n.*

While there always exists a TT decomposition for any arbitrary tensor, much like matrix decompositions these TT decompositions are not unique. Multiplying each slice $\boldsymbol{\mathcal{U}}_n(:, i_n, :)$ on the right by an invertible matrix $\boldsymbol{B}$ and each slice $\boldsymbol{\mathcal{U}}_{n+1}(:, i_{n+1}, :)$ on the left by $\boldsymbol{B}^{-1}$ yields a new TT decomposition. However, this lack of uniqueness will prove very useful later as it will allow us to reduce the TT ranks of certain tensors.

An outline of the proof of Theorem 3.2.8 goes as follows; Consider an Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ and suppose that $\text{rank}(\boldsymbol{X}^{(n)}) = R_n$ for each $n \in \{1, \ldots, N\}$. The unfolding $\boldsymbol{X}^{(1)}$ has rank R_1, by selection, so it admits a singular value decomposition

$$\boldsymbol{X}^{(1)} = \boldsymbol{U}_1 V_1^T$$

with $\boldsymbol{U}_1 \in \mathbb{R}^{I_1 \times R_1}$ and $\boldsymbol{V}_1 \in \mathbb{R}^{R_1 \times (I_2 \ldots I_N)}$. Viewing the matrix $\boldsymbol{U}_1$ as a tensor $\boldsymbol{\mathcal{U}}_1 \in \mathbb{R}^{1 \times I_1 \times R_1}$ gives the first core in the TT decomposition. We can also view $\boldsymbol{V}_1$ as an Nth-order tensor $\boldsymbol{\mathcal{V}}_1$ with mode lengths given by $R_1, I_2, \ldots, I_N$. In index form $\boldsymbol{\mathcal{X}}$ is then given by

$$\boldsymbol{\mathcal{X}}(i_1, \ldots, i_2) = \sum_{r_1=1}^{R_1} \boldsymbol{U}(i_1, r_1) \boldsymbol{\mathcal{V}}_1(r_1, i_2, \ldots, i_N).$$

It is shown in the proof of Theorem 3.2.8 that

$$\text{rank}(\boldsymbol{V}_1^{(n)}) \leq R_n$$

for each $n \in \{2, \ldots, N\}$. We can form $\boldsymbol{V}_1^{(2)}$ by combining r_1 and i_2 into one long index $(r_1 i_2)$ and combining $i_3, \ldots, i_N$ into another long index $(i_3 \ldots i_N)$. Since $\text{rank}(\boldsymbol{V}_1^{(2)}) \leq R_2$, it admits a singular value decomposition given in index form by

$$\boldsymbol{V}_1^{(2)}(r_1, i_2 | i_3, \ldots, i_N) = \sum_{r_2=1}^{R_2} \boldsymbol{U}_2(r_1, i_2 | r_2) \boldsymbol{V}_2(r_2 | i_3, \ldots, i_N).$$

By separating r_1 and i_2 we can turn $\boldsymbol{U}_2$ into a third-order tensor $\boldsymbol{\mathcal{U}}_2 \in \mathbb{R}^{R_1 \times I_2 \times R_2}$. This is the second core of the decomposition. Continuing in this fashion with $\boldsymbol{V}_2$ and the subsequent matrices $\boldsymbol{V}_n$ up to $n = N$ will yield the cores $\boldsymbol{\mathcal{U}}_n$ of the TT decomposition. Since the matrix decompositions in this procedure are calculated using singular value decompositions, we call this method of computing the TT decomposition process the *TT-SVD*.

In practice it is unlikely that the unfolding matrices $\boldsymbol{X}^{(n)}$ will be of low rank. Usually the best that we can do is have matrices well approximated by low rank matrices. If instead we compute the unfolding matrices using a best rank-R_n approximation according to the SVD we can compute an approximate TT decomposition.

Theorem 3.2.9 (Theorem 2.2 in [44])
Let $\boldsymbol{\mathcal{X}}$ be an Nth-order tensor and suppose that for each $n = 1, \ldots, N$ the unfolding matrix $\boldsymbol{X}^{(n)}$ is only approximately low rank. That is,

$$\boldsymbol{X}^{(n)} = \boldsymbol{R}^{(n)} + \boldsymbol{E}^{(n)}, \quad \textit{rank}(\boldsymbol{R}^{(n)}) = R_n, \quad ||\boldsymbol{E}^{(n)}|| = \varepsilon_n,$$

for some small integer R_n and small positive real number ε. Then the TT-SVD results in a tensor $\boldsymbol{\mathcal{Y}}$ in TT format with TT ranks given by r_n for each n. Moreover, the error is bounded by

$$||\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}||_F \leq \left(\sum_{n=1}^{N-1} \varepsilon_n^2\right)^{1/2}.$$

Theorem 3.2.9 implies that the TT SVD can be used to approximate an Nth-order tensor $\boldsymbol{\mathcal{X}}$ with error less than ε by approximating the unfolding matrices with an accuracy of at least $\varepsilon(N-1)^{-1/2}$. In fact if we approximate the unfolding matrices with an accuracy $\varepsilon||\boldsymbol{\mathcal{X}}||_F(N-1)^{-1/2}$ then the relative error in the approximation will be bounded above by ε. This can be done by truncating the SVD's of the unfolding matrices at the required accuracy. In this case the TT ranks will be the δ-ranks of the unfolding matrices. By δ-rank of a matrix $\boldsymbol{A}$ we mean the minimum rank possible for a matrix $\boldsymbol{B}$ satisfying $||\boldsymbol{A} - \boldsymbol{B}|| \leq \delta$. We denote the δ-rank of $\boldsymbol{A}$ by $rank_\delta(\boldsymbol{A})$.

Another important result that follows from Theorem 3.2.9 is that a best rank-R approximation always exist, in the sense that the TT ranks are bounded by R.

Corollary 3.2.10 (Corollary 2.4 in [44].)
Let $\boldsymbol{\mathcal{X}}$ be any Nth-order tensor and pick integers $R_1, \ldots, R_{N-1}$. Then there always exists a best approximation to $\boldsymbol{\mathcal{X}}$ under the Frobenius norm with TT ranks bounded by R_n for each $n = 1, \ldots, N-1$ (Denote this approximation by $\boldsymbol{\mathcal{X}}^{best}$). Furthermore, the approximation $\boldsymbol{\mathcal{Y}}$ given by the TT-SVD is quasi-optimal in the sense that

$$||\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}||_F \leq (d-1)^{1/2}||\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}}^{best}||_F.$$

Algorithm 1 (Algorithm 1 in [44]) gives a formal description of using the TT-SVD to approximate any tensor to any accuracy with a tensor in TT format.

3.2.2.1 Algebraic Operations in TT format

It is not often that we wish to compute a tensor and then do nothing with it. We usually aim to perform calculations with this tensor. Calculations such as adding it to another tensor, computing a Hadamard product, or calculating its n-mode product with a matrix or vector. The number of floating point operations that need to be performed to compute the operations grow exponentially with the number of modes if done with the full tensor. However, in the TT format these calculations can be done with a computational cost that scales linearly with the number of modes. Moreover, these operations can be done in such a way that the result is also given in TT format.

Proposition 3.2.11 (Adapted from discussion in Section 4.1 of [44].)
Suppose we have two Nth-order tensors $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ given in TT format. Let $\boldsymbol{\mathcal{X}}$ have cores denoted by $\boldsymbol{\mathcal{X}}_1, \ldots, \boldsymbol{\mathcal{X}}_N$ and TT ranks denoted by $R_1^X, \ldots, R_N^X$. Similarly, let $\boldsymbol{\mathcal{Y}}$ have cores denoted by $\boldsymbol{\mathcal{Y}}_1, \ldots, \boldsymbol{\mathcal{Y}}_N$ and TT ranks denoted by $R_1^Y, \ldots, R_N^Y$.

1. *For any scalar $\lambda \in \mathbb{R}$, the tensors $\boldsymbol{\mathcal{Z}} := \lambda\boldsymbol{\mathcal{X}}$ has a TT decomposition with the same TT ranks as $\boldsymbol{\mathcal{X}}$ and its cores are given by*

$$\widehat{\boldsymbol{\mathcal{X}}}_n := \begin{cases} \boldsymbol{\mathcal{X}}_n & \text{if } n \neq m, \\ \lambda\boldsymbol{\mathcal{X}}_m & \text{if } n = m, \end{cases}$$

where m labels the mode that minimises $R_{m-1}I_mR_m$. Therefore, scalar multiplication in the TT format requires at most $\mathcal{O}(IR^2)$ operations.

2. *The tensor* $\boldsymbol{\mathcal{Z}} := \boldsymbol{\mathcal{X}} + \boldsymbol{\mathcal{Y}}$ *has a TT decomposition with ranks* $R_n^Z = R_n^X + R_n^Y$ *and cores* $\boldsymbol{\mathcal{Z}}_n$ *defined by*

$$\boldsymbol{\mathcal{Z}}_n(:, i_n, :) = \begin{bmatrix} \boldsymbol{\mathcal{X}}_n(:, i_n, :) & 0 \\ 0 & \boldsymbol{\mathcal{Y}}_n(:, i_n, :) \end{bmatrix} \quad \textit{for} \quad n \in \{2, \dots, N-1\}$$

and

$$\boldsymbol{\mathcal{Z}}_1(i_1, :) = \begin{bmatrix} \boldsymbol{\mathcal{X}}_1(i_1, :) & \boldsymbol{\mathcal{Y}}_1(i_1, :) \end{bmatrix}, \quad \boldsymbol{\mathcal{Z}}_N(:, i_N) = \begin{bmatrix} \boldsymbol{\mathcal{X}}_N(:, i_N) \\ \boldsymbol{\mathcal{Y}}_N(:, i_N) \end{bmatrix}.$$

3. *The Hadamard product* $\boldsymbol{\mathcal{Z}} := \boldsymbol{\mathcal{X}} * \boldsymbol{\mathcal{Y}}$ *has a TT decomposition with ranks* $R_n^Z = R_n^X R_n^Y$ *and cores* $\boldsymbol{\mathcal{Z}}_n$ *defined by*

$$\boldsymbol{\mathcal{Z}}_n(:, i_n, :) = \boldsymbol{\mathcal{X}}_n(:, i_n, :) \otimes \boldsymbol{\mathcal{Y}}(:, i_n, :) \quad \textit{for} \quad n \in \{1, \dots, N\}.$$

It then follows that computing the Hadamard product in TT format requires $\mathcal{O}(NIR^4)$ *operations.*

4. *Pick any* $n \in \{1, \dots, N\}$ *and let* $\boldsymbol{M} \in \mathbb{R}^{J \times I_n}$. *Then the n-mode product* $\boldsymbol{\mathcal{Z}} := \boldsymbol{\mathcal{X}} \times_n \boldsymbol{M}$ *has a TT decomposition with ranks* $R_1^X, \dots, R_N^X$ *and cores given by*

$$\boldsymbol{\mathcal{Z}}_l = \boldsymbol{\mathcal{X}}_l, \quad \textit{for all} \quad l \neq n,$$

and

$$\boldsymbol{\mathcal{Z}}_n(r_{n-1}, :, :) = \boldsymbol{M}\boldsymbol{\mathcal{X}}(r_{n-1}, :, :), \quad \textit{for} \quad r_{n-1} = 1, \dots, R_{n-1}.$$

It then follows that computing the n-mode product requires $\mathcal{O}(JIR^2)$ *operations.*

5. *Define the family of matrices* $\boldsymbol{\Gamma}_n$ *by*

$$\boldsymbol{\Gamma}_n = \sum_{i_n}^{I_n} \boldsymbol{\mathcal{X}}_n(:, i_n, :) \otimes \boldsymbol{\mathcal{Y}}_n(:, i_n, :).$$

Then the inner product $\langle \boldsymbol{\mathcal{X}} | \boldsymbol{\mathcal{Y}} \rangle$ *is given by the recurrence*

$$\boldsymbol{v}_n = \boldsymbol{v}_{n-1}\boldsymbol{\Gamma}_n, \quad n \in \{2, \dots, N\}$$

and

$$\boldsymbol{v_1} = \boldsymbol{\Gamma}_1.$$

It then follows that computing the inner product requires $\mathcal{O}(NIR^3)$ *operations.*

The proofs of all of these propositions can be found in Section 4 of [44], along with formal algorithms for their implementation. A proof for Proposition 3.2.11.4 is not given in [44] but we provide a proof in the appendix, see Proposition A.0.1 and Algorithm A1 for a formal algorithm of its implementation.

Remark 3.2.12
The operation `reshape` is the reshaping command from Matlab and `numel` is the number of elements in a tensor.

Algorithm 1: TT-SVD

Input: An Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$, desired relative error ε.

Result: TT cores $\boldsymbol{\mathcal{U}}_1, \ldots, \boldsymbol{\mathcal{U}}_N$ of a TT approximation $\boldsymbol{\mathcal{Y}}$ to a tensor $\boldsymbol{\mathcal{X}}$. The tensor $\boldsymbol{\mathcal{Y}}$ has TT ranks R_n equal to the δ-ranks of the unfoldings $\boldsymbol{X}^{(n)}$, with $\delta = \varepsilon ||\boldsymbol{\mathcal{X}}||_F (N-1)^{-1/2}$. The result $\boldsymbol{\mathcal{Y}}$ satisfies

$$||\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}||_F \leq \varepsilon ||\boldsymbol{\mathcal{X}}||_F.$$

Initialisation:

Set the truncation parameter $\delta = \varepsilon ||\boldsymbol{\mathcal{X}}||_F (N-1)^{-1/2}$.
Set a temporary tensor $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}}$.
Set $R_0 = 1$.

Main Algorithm:

for $n = 1, \ldots, N$ **do**
  $\boldsymbol{\mathcal{Z}} := \texttt{reshape}\Big(Z, \Big[R_{n-1} I_N, \frac{\texttt{numel}(\boldsymbol{\mathcal{Z}})}{R_{n-1} I_N}\Big]\Big)$.
  Compute δ-truncated SVD: $\boldsymbol{\mathcal{Z}} = \boldsymbol{U}\boldsymbol{V}^T + \boldsymbol{E}$,$||E||_F \leq \delta$ and $R_n = \text{rank}_\delta(\boldsymbol{\mathcal{Z}})$.
  Set new core $\boldsymbol{\mathcal{U}}_n = \texttt{reshape}(\boldsymbol{U}, [R_{n-1}, I_n, R_n)$.
  $\boldsymbol{\mathcal{Z}} = \boldsymbol{V}^T$.
end
$\boldsymbol{\mathcal{U}}_N = \boldsymbol{\mathcal{Z}}$.
Return tensor $\boldsymbol{\mathcal{Y}}$ in TT format with cores $\boldsymbol{\mathcal{U}}_1, \ldots, \boldsymbol{\mathcal{U}}_N$.

3.2.2.2 Rounding in TT format

The five operations in Proposition 3.2.11 are the most common algebraic operations that are used when working with the TT format, both analytically and computationally. However, both addition and the Hadamard product increase the rank of the TT decomposition. This can be very dangerous since the computational cost of these operations in TT format depends heavily on the rank of the operand tensors. We can use the the lack of uniqueness of TT decompositions to our advantage to prevent rank growth. Section 3 of [44] is dedicated to the derivation of a *rounding* procedure for the TT format, in which the ranks are systematically reduced by singular value decompositions of certain auxiliary matrices. We give only an outline of the derivation and the formal algorithm is given in Algorithm 2.

For an arbitrary Nth-order tensor $\boldsymbol{\mathcal{X}}$ in TT format with ranks R_n and cores $\boldsymbol{\mathcal{X}}_n$, the rounding algorithm works by performing a single right-to-left sweep of the cores to orthogonalise the rows of the cores $\boldsymbol{\mathcal{X}}_n$ (by 'rows' we mean the vectors $\boldsymbol{\mathcal{X}}_n(r_{n-1}, i_n, :)$ for each $r_{n-1} = 1 \ldots R_{n-1}$ and $i_n = 1, \ldots, I_n$). Starting with the last core $\boldsymbol{\mathcal{X}}_N$, which can be viewed as an $R_{n-1} \times I_N$ matrix. Orthogonalise its rows to find a QR-decomposition

$$\boldsymbol{\mathcal{X}}_N(:, i_N) = \boldsymbol{R}_{N-1} \boldsymbol{\mathcal{Q}}(:, i_N),$$

where $\boldsymbol{\mathcal{Q}}(:, i_N)$ has orthonormal rows. Then define a new tensor $\boldsymbol{\mathcal{X}}'_{N-1}$ by

$$\boldsymbol{\mathcal{X}}'_{N-1}(:, i_{n-1}, :) = \boldsymbol{\mathcal{X}}_{N-1}(:, i_{n-1}, :) \boldsymbol{R}_N,$$

it follows from the definition of the n-mode product that this is equivalent to

$$\boldsymbol{\mathcal{X}}'_{N-1} = \boldsymbol{\mathcal{X}}_{N-1} \times_3 \boldsymbol{R}_N.$$

It then follows that

$$\boldsymbol{\mathcal{X}}(i_1,\ldots,i_N) = \boldsymbol{\mathcal{X}}_1(i_1,:)\ldots\boldsymbol{\mathcal{X}}_{N-1}(:,i_{N-1},:)\boldsymbol{R}_N\boldsymbol{\mathcal{Q}}_N(:,i_N).$$

Viewing $\boldsymbol{\mathcal{X}}'_{N-1}$ as an $R_{N-2}\times(I_{N-1}R_{N-1})$ matrix, orthogonalise its rows so that

$$\boldsymbol{\mathcal{X}}'_{N-1}(:,i_{N-1},:) = \boldsymbol{R}_{N-1}\boldsymbol{\mathcal{Q}}_{N-1}(:,i_{N-1},:) \tag{3.10}$$

where $\boldsymbol{\mathcal{Q}}(:,i_{N-1},:)$ has orthonormal rows. By defining the $R_{n-1}\times I_n\times R_n$ tensors

$$\boldsymbol{\mathcal{X}}'_{n-1} := \boldsymbol{\mathcal{X}}_n \times_3 \boldsymbol{R}_n, \quad n = N-1,\ldots,2$$

continue this process until we have

$$\boldsymbol{\mathcal{X}}(i_1,\ldots,i_N) = \boldsymbol{\mathcal{X}}'_1(i_1,:)\boldsymbol{\mathcal{Q}}_2(:,i_2,:)\ldots\boldsymbol{\mathcal{Q}}_N(:,i_N,:)$$

such that $\boldsymbol{\mathcal{Q}}_n(:,i_n,:)$ has orthonormal rows for each $i_n\in\{1,\ldots,I_n\}$ for all $n\in\{2,\ldots,N\}$. This process of orthogonalising the rows of the unfolding matrices of the cores as in (3.10) is denoted by $QR_{\text{rows}}(\boldsymbol{\mathcal{X}}_n)$ and its outputs are the matrix $\boldsymbol{R}_n\in R^{R_{n-1}\times R_{n-1}}$ and the third-order tensor $\boldsymbol{\mathcal{Q}}_n\in\mathbb{R}^{R_{n-1}\times I_n\times R_n}$.

After the orthogonalisation of the cores a left-to-right sweep of the cores is performed to reduce the TT ranks. The rank reduction works by performing reduced SVD's on certain unfoldings of the core tensors. If $\boldsymbol{\mathcal{X}}'_1$ has matrix rank R'_1 then by a reduced SVD we can write

$$\boldsymbol{\mathcal{X}}'_1 = \boldsymbol{\mathcal{U}}_1\boldsymbol{\Lambda}_1\boldsymbol{V}_1^T$$

where $\boldsymbol{\mathcal{U}}_1$ is an $I_1\times R'_1$ matrix with orthonormal rows, $\boldsymbol{\Lambda}_1$ is an $R'_1\times R'_1$ diagonal matrix and $\boldsymbol{V}_1$ is a $R_1\times R'_1$ matrix. The entries of $\boldsymbol{\mathcal{X}}$ are now given by

$$\begin{aligned}\boldsymbol{\mathcal{X}}(i_1,\ldots,i_N) &= \boldsymbol{\mathcal{U}}_1(i_1,:)(\boldsymbol{\Lambda}_1\boldsymbol{V}_1)^T\boldsymbol{\mathcal{Q}}_2(:,i_2,:)\ldots\boldsymbol{\mathcal{Q}}_N(:,i_N)\\ &= \boldsymbol{\mathcal{U}}_1(i_1,:)\boldsymbol{\mathcal{G}}_2(:,i_2,:)\ldots\boldsymbol{\mathcal{Q}}_N(:,i_N,:),\end{aligned}$$

where we defined

$$\boldsymbol{\mathcal{G}}_2 = \boldsymbol{\mathcal{Q}}_2\times_1(\boldsymbol{\Lambda}_1\boldsymbol{V}_1)^T,$$

which is an $R'_1\times I_2\times R_2$ tensor. Viewing $\boldsymbol{\mathcal{G}}_2$ as an $(R'_1I_2)\times R_2$ with matrix rank R'_2, we perform a reduced SVD to find

$$\boldsymbol{\mathcal{G}}_2(:,I_2,:) = \boldsymbol{\mathcal{U}}_2(:,i_2,:)\boldsymbol{\Lambda}_2\boldsymbol{V}_2^T,\ \forall\ i_2\{1,\ldots,I_2\}.$$

Continue this process by defining the $R'_{n-1}\times I_n\times R_n$ tensors

$$\boldsymbol{\mathcal{G}}_{n+1} = \boldsymbol{\mathcal{Q}}_{n+1}\times_1(\boldsymbol{\Lambda}_n\boldsymbol{V}_n)^T, \quad n = 3,\ldots,N$$

until we find

$$\boldsymbol{\mathcal{X}}(i_1,\ldots,i_N) = \boldsymbol{\mathcal{U}}(i_1,:)\boldsymbol{\mathcal{U}}_2(:,i_2,:)\ldots\boldsymbol{\mathcal{G}}_N(:,i_N,:).$$

Renaming $\boldsymbol{\mathcal{U}}_N := \boldsymbol{\mathcal{G}}_N$, gives a new TT decomposition of $\boldsymbol{\mathcal{X}}$ with new cores $\boldsymbol{\mathcal{U}}_n$ and ranks R'_n for $n\in\{1,\ldots,N\}$. These new TT ranks R_{n-1} are the matrix ranks found by viewing the tensors $\boldsymbol{\mathcal{G}}_n$ as $R'_{n-1}\times(I_nR_n)$ matrices. If the reduced SVD's are only approximately correct with accuracy δ, then the new ranks are the δ-ranks of the unfoldings of $\boldsymbol{\mathcal{G}}_n$. It is also shown in [44] that these ranks are the δ-ranks of the unfoldings $\boldsymbol{\mathcal{X}}^{(n)}$ for each n. By

construction, the approximation formed by the rounding process satisfies the same error bounds as the TT-SVD given in Theorem 3.2.9. In Algorithm 2 we use "$\text{SVD}_\delta(\boldsymbol{M})$" to denote a function that computes a δ-truncated SVD and returns the factor matrices $\boldsymbol{U}, \boldsymbol{\Lambda}$ and $\boldsymbol{V}$ as well as the δ-rank of $\boldsymbol{M}$.

In the orthogonalisation sweep the QR-decompositions of the $R_{n-1} \times (I_n R_n)$ matrices require $\mathcal{O}(IR^3)$ operations, there are N of these to compute. The compression sweep requires the SVD of N matrices that are $(R_{n-1}I_n) \times R_n$, which also requires $\mathcal{O}(IR^3)$ operations. So, in total the rounding procedure requires $\mathcal{O}(NIR^3)$ operations.

Algorithm 2: TT-rounding

Input: An Nth-order tensor $\boldsymbol{\mathcal{X}}$ and a desired accuracy ε.

Result: $\boldsymbol{\mathcal{Y}}$ in the TT format with TT ranks R'_n equal to the δ-ranks of the unfoldings $\boldsymbol{\mathcal{X}}^{(1)}$, where $\delta = \varepsilon ||\boldsymbol{\mathcal{X}}||_F (N-1)^{-1/2}$. The computed approximation satisfies

$$||\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}||_F \leq \varepsilon ||\boldsymbol{\mathcal{X}}||_F.$$

Initialisation:

Let $\boldsymbol{\mathcal{X}}_1, \ldots, \boldsymbol{\mathcal{X}}_N$ be the TT cores and $R_0, \ldots, R_N$ the TT ranks of $\boldsymbol{\mathcal{X}}$.

Main algorithm:

{Right-to-left orthogonalisation}

for $n = N, \ldots, 2$ **do**

  $[\boldsymbol{R}_n, \boldsymbol{\mathcal{Q}}_n] = QR_{\text{rows}}(\boldsymbol{\mathcal{X}}_n)$.

  $\boldsymbol{\mathcal{X}}'_{n-1} := \boldsymbol{\mathcal{X}}_n \times_3 \boldsymbol{R}_n$.

end

Set $\boldsymbol{\mathcal{G}}_1 := \boldsymbol{\mathcal{X}}'_1$.

Set $R'_0 = 1$.

{Rank reduction of Orthogonalised representation}

for $n = 1, \ldots, N-1$ **do**

  {Compute δ-truncated SVD}

  $\boldsymbol{G}_n := \texttt{reshape}(\boldsymbol{\mathcal{G}}_n, [R'_{n-1}I_n, R_n])$.

  $[\boldsymbol{\mathcal{U}}_n, \boldsymbol{\Lambda}, \boldsymbol{V}, R'_n] := \text{SVD}_\delta(\boldsymbol{G}_n)$.

  $\boldsymbol{\mathcal{G}}_{n+1} := \boldsymbol{\mathcal{Q}}_{n+1} \times_1 (\boldsymbol{\Lambda}\boldsymbol{V})^T$.

end

$\boldsymbol{\mathcal{U}}_n := \boldsymbol{\mathcal{G}}_N$.

Return: $\boldsymbol{\mathcal{Y}}$ in TT format with cores $\boldsymbol{\mathcal{U}}_1, \ldots, \boldsymbol{\mathcal{U}}_N$.

3.2.2.3 Cross approximation of Tensors

In higher-dimensional problems it is often not possible to store a tensor in its full format, but the TT-SVD requires the full tensor to calculate a TT decomposition. However, if we can represent the target tensor with some kind of rule or formula that generates the entries then it is possible to create a TT decomposition without evaluating all of the entries. This is made possible by the TT-cross approximation method introduced by Oseledets and Tyrtyshnikov in [43].

The basic idea of the TT-cross method is to replace the reduced-SVD's in the TT-SVD process by skeleton decompositions (see [26, Sec. 3]). By using the skeleton decomposition, it is possible to create a TT decomposition without ever storing the full tensor. Only the exact case is considered in [43], but Savostyanov shows in [51] that this TT cross method is quasi-optimal in some sense.

So consider an Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ and assume its first unfolding matrix $\boldsymbol{X}^{(1)}$ has matrix rank R_1. Suppose we we know a set of indices $\mathcal{J}_1$ of R_1 linearly independent columns of $\boldsymbol{X}^{(1)}$. We know that these columns exist because $\boldsymbol{X}^{(1)}$ has matrix rank R_1. The columns of $\boldsymbol{X}^{(1)}$ are denoted by $(i_2, \ldots, i_N)$ as one long index, so the elements of $\mathcal{J}_1$ are;

$$(i_2^{(r_1)}, \ldots, i_N^{(r_1)}), \quad \text{for} \quad r_1 \in \{1, \ldots, R_1\}.$$

Define a matrix $\boldsymbol{C}_1 \in \mathbb{R}^{I_1 \times R_1}$ by the columns specified by $\mathcal{J}_1$, that is

$$\boldsymbol{C}_1(i_1, r_1) = \boldsymbol{\mathcal{X}}(i_1, i_2^{(r_1)}, \ldots, i_N^{(r_1)}).$$

Now let $\widehat{\boldsymbol{X}_1}$ be an $R_1 \times R_1$ sub-matrix of $\boldsymbol{C}_1$ with maximal absolute determinant (also called *matrix volume*) and rows specified by an index set $\mathcal{I}_1$ with elements

$$i_1^{(r_1)} \quad \text{for} \quad r_1 \in \{1, \ldots, R_1\}.$$

A quasi-maximum volume matrix can be found in $\mathcal{O}(IR^2)$ operations by the *maxvol* algorithm from [61] and [14]. Define $\boldsymbol{R}_1$ to be the unfolding $\boldsymbol{R}_1^{(1)}$ of the $R_1 \times I_2 \times \ldots \times I_N$ tensor $\boldsymbol{\mathcal{R}}_1$ defined by

$$\boldsymbol{\mathcal{R}}_1(r_1, i_2, \ldots, i_N) = \boldsymbol{\mathcal{X}}(i_1^{(r_1)}, i_2, \ldots, i_N).$$

The matrix $\boldsymbol{R}_1$ then corresponds to the rows of $\boldsymbol{X}^{(1)}$ that contain $\widehat{\boldsymbol{X}_1}$. A skeleton decomposition of $\boldsymbol{X}^{(1)}$ is then given by

$$\boldsymbol{X}^{(1)} = \boldsymbol{C}_1 \widehat{\boldsymbol{X}}_1^{-1} \boldsymbol{R_1}.$$

As with the TT-SVD, the first TT core is taken to be

$$\boldsymbol{\mathcal{U}}_1 = \boldsymbol{C}_1 \widehat{\boldsymbol{X}}_1^{-1}. \tag{3.11}$$

Since $\boldsymbol{\mathcal{R}}_1$ is a sub-tensor of $\boldsymbol{\mathcal{X}}$, it follows that for each $n \in \{2, \ldots, N\}$ if $\boldsymbol{X}^{(n)}$ has matrix rank R_n, then the matrix rank of the unfolding $\boldsymbol{R}_1^{(n)}$ is bounded above by R_n. Consider the unfolding $\boldsymbol{R}_1^{(2)}$ and suppose we know $\mathcal{J}_2$ to be an index set with elements

$$(i_3^{(r_2)}, \ldots, i_N^{(r_2)}), \quad \text{for} \quad r_2 \in \{1, \ldots, R_2\}.$$

which denote the columns of $\boldsymbol{X}^{(2)}$ containing an $R_2 \times R_2$ quasi-maximum volume sub-matrix of $\boldsymbol{\mathcal{R}}_1^{(2)}$. The matrix $\boldsymbol{C}_2$ defined by these columns has size $(R_1 I_2) \times R_2$ and has entries given by

$$\boldsymbol{C}_2(r_1, i_2|r_2) = \boldsymbol{\mathcal{R}}_1(r_1, i_2|i_3^{(r_2)}, \ldots, i_N^{(r_2)}) = \boldsymbol{\mathcal{X}}(i_1^{(r_1)}, i_2|i_3^{(r_2)}, \ldots, i_N^{(r_2)}),$$

where the indices $i_1^{(r_1)}$ are from $\mathcal{I}_1$ above. Using the maxvol procedure find an $R_2 \times R_2$ sub-matrix $\widehat{\boldsymbol{X}}_2$ of $\boldsymbol{C}_2$ with quasi-maximum volume. Let the rows of $\widehat{\boldsymbol{X}}_2$ be specified by an index set $\mathcal{I}_2$ with elements

$$(i_1^{(r_2)}, i_2^{(r_2)}) \text{ for } r_2 \in \{1, \ldots, R_2\}.$$

Define $\boldsymbol{R}_2$ to be the unfolding $\boldsymbol{R}_2^{(1)}$ of the $R_2 \times I_3 \times \ldots \times I_N$ tensor $\boldsymbol{\mathcal{R}}_2$ defined by

$$\boldsymbol{\mathcal{R}}_2(r_2, i_3, \ldots, i_N) = \boldsymbol{\mathcal{X}}(i_1^{(r_1)}, i_2^{(r_2)}, i_3, \ldots, i_N).$$

The next TT core $\boldsymbol{\mathcal{U}}_2$ is taken such that

$$\boldsymbol{U}_2^{(2)} = \boldsymbol{C}_2 \widehat{\boldsymbol{X}}_2^{-1}. \tag{3.12}$$

We continue in this fashion until we have $\boldsymbol{\mathcal{U}}_n \in R^{R_{n-1} \times I_n \times R_n}$ defined analogously to (3.12) for each $n \in \{1, \ldots, N-1\}$ and a tensor $\boldsymbol{\mathcal{R}}_N \in \mathbb{R}^{R_{N-1} \times I_N}$. The final core is then taken as $\boldsymbol{\mathcal{U}}_N := \boldsymbol{\mathcal{R}}_N$. It should be noted that since we are only finding quasi-maximal volume sub-matrices, this decomposition is not exact.

If for each $n \in \{1, \ldots, N\}$ we know the column index set $\mathcal{J}_n$ corresponding to columns of $\boldsymbol{X}^{(n)}$ containing an $R_n \times R_n$ quasi-maximal volume sub-matrix of the unfolding $\boldsymbol{X}^{(n)}$, then computing the matrix $\boldsymbol{C}_n$ requires only $R_{n-1} I_n R_n$ entries of $\boldsymbol{\mathcal{X}}$. So in total we only need to draw $\mathcal{O}(NIR^2)$ entries of $\boldsymbol{\mathcal{X}}$. In order to complete the construction of the cores we need to perform the maxvol algorithm N times, which requires $\mathcal{O}(IR^3)$ operations. If the maximum number of operations required to evaluate a single entry in $\boldsymbol{\mathcal{X}}$ is C_X then the TT-cross procedure requires at most $\mathcal{O}(C_X NIR^2) + \mathcal{O}(NIR^3)$ operations.

In practice it will hardly ever be known what the index sets $\mathcal{J}_n$ are, so they must be computed somehow. An alternating row-column algorithm is given in [6]. The algorithm presented there performs the TT-cross procedure above first, to find the index sets for the rows. Then it performs an analogous procedure backwards (starting with the unfolding $\boldsymbol{\mathcal{X}}^{(N)}$) to find new index sets for the columns. This alternating pattern is repeated until the desired accuracy is met. Algorithm 1 in [6] gives a formal description of this procedure and there it is shown to require $\mathcal{O}(NIR^2)$ entries of $\boldsymbol{\mathcal{X}}$ and a further $\mathcal{O}(NIR^3)$ operations. The Matlab package TT-toolbox([45]) implements a version of this procedure in the function `amen_cross`.

We have covered all the fundamental results on tensors that we will be utilising in this thesis. However, this was by no means an exhaustive overview of tensors and tensor decompositions. There are many more tensor decompositions such as the *Tucker decomposition* (see [60],[58],[59],[4] and[28]). The tucker decomposition tries to approximate an Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ by an Nth-order core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times \ldots \times R_N}$ and factor matrices $\boldsymbol{M}_n \in \mathbb{R}^{I_n \times R_n}$ for $n \in \{1, \ldots, N\}$ such that

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{M}_1 \ldots \times_N \boldsymbol{M}_N.$$

The core tensor $\boldsymbol{\mathcal{G}}$ still needs to be stored in full, which means that the number of values that needs to be stored still grows exponentially with the order N. In total the Tucker Decomposition requires the storage of $\mathcal{O}(R^N + NIR)$ values. For this reason, especially for our purposes, the TT decomposition is seen as superior. The tensor train and the Tucker decompositions are the most successful special cases of a more general decomposition known as a *Hierarchical Tree decomposition*, a *Hierarchical-Tucker decomposition* or a *Tree-Tucker decomposition* (see [46],[16] and [15]).

Chapter 4

A Finite Volume Method for Convection-Diffusion Equations

Our main objective in this thesis is to show that tensor trains can be used to speed up numerical approximation of PDEs. This goal is split into two parts. In this chapter we develop a finite volume method for a particular PDE. Once a stable numerical scheme is developed how to implement this scheme using tensors and TT decompositions is demonstrated in the following chapter.

We will create a numerical scheme for a general class of PDEs, in particular those given by the general *convection-diffusion equation* (also known as the *advection-diffusion equation*).

Definition 4.0.1
Let $\Omega \subset \mathbb{R}^D$ be an open set. A function $\phi(t, \boldsymbol{x})$ satisfies the *general convection-diffusion equation* on Ω if for all $t > 0$ and $\boldsymbol{x} \in \Omega$ we have

$$\frac{\partial \phi}{\partial t}(t, \boldsymbol{x}) - \nabla \cdot [\boldsymbol{w}(t, \boldsymbol{x}) * \nabla \phi(t, \boldsymbol{x})] + \nabla \cdot [\phi(t, \boldsymbol{x}) \boldsymbol{v}(t, \boldsymbol{x})] = s(t, \boldsymbol{x}), \tag{4.1}$$

where $\boldsymbol{v}$ and $\boldsymbol{w}$ are D-dimensional vector fields called the *convection-velocity field* and the *diffusion coefficient*, respectively. In particular, $\boldsymbol{w}$ is a positive vector field, i.e., all its components are always positive. The operation $*$ denotes the Hadamard product. The function s is a scalar function called the *source term*. When $\boldsymbol{w}, \boldsymbol{v}$ and s are all constant in t we say that (4.1) is *time homogeneous*.

Remark 4.0.2
We will refer to the variables t and $\boldsymbol{x}$ as "time" and "space" respectively. However, this does not suggest that the convection-diffusion equation relates only to space-time phenomena.

General convection-diffusion equations appear in many applications. These applications include fluid mechanics where ϕ may represent chemical concentration in a river [3], in the study of semiconductors it is called the *drift-diffusion equation* [35, Chap. 3]. They also appear in mathematical finance as the *Black-Scholes equation* where ϕ would represent the price of an *option* [50, Chap. 5-6]

In the study of diffusion processes the convection-diffusion equation with no source appears as the *Kolmogorov forward equation*. Here $\phi(t, \boldsymbol{x})$ represents the probability distribution over the state space of the process given the initial conditions $\phi(0, \boldsymbol{x})$, [31, Sect. 15.1],

[55, Chap. 3]. These are the type of convection diffusion equations that we will consider in the remainder of this thesis. So, from now on ϕ is assumed to be a probability density, i.e., the integral of ϕ over Ω is 1 and it is strictly non-negative for all $t > 0$. Only time homogeneous equations will be considered here. However, with some extra work one could extend the method presented here to time-dependent convection velocities and diffusion coefficients.

In summary, we will create a positivity and integral preserving discrete approximation to the following general initial value problem;

$$\begin{cases} \frac{\partial \phi}{\partial t}(t, \boldsymbol{x}) - \nabla \cdot [\boldsymbol{w}(\boldsymbol{x}) * \nabla \phi(t, \boldsymbol{x})] + \nabla \cdot [\phi(t, \boldsymbol{x}) \boldsymbol{v}(\boldsymbol{x})] = 0 & \forall \boldsymbol{x} \in \mathbb{R}^D,\ t > 0 \\ \phi(0, \boldsymbol{x}) = \phi_0(\boldsymbol{x}), & \forall \boldsymbol{x} \in \mathbb{R}^D. \end{cases} \tag{4.2}$$

By not considering a general bounded domain we save ourselves from being restricted to a specific boundary condition problem. This allows for the creation of a method that can be easily specialised to accommodate various boundary conditions. How to impose boundary conditions is illustrated in examples.

4.1 Derivation of the Finite Volume Method

We derive our finite volume method by following the derivation in [40] by Fox et al. They consider the *continuity equation*, which is simply the time homogeneous convection-diffusion equation with the diffusion coefficient set to zero. Therefore, our method is an extension of their method in one respect. However, Fox et al. considered arbitrary control volumes while only rectangular control volumes are considered here. It should also be noted that Fox et al.'s finite volume method is a specialisation of Mertlet and Vovelle's method for time inhomogeneous equations from [38]. By following their derivations one could extend our method to arbitrary control volumes and time inhomogeneous equations.

As usual with the finite volume method, start by defining the control volumes.

Definition 4.1.1
Fix a time step $\Delta t > 0$ and for each $d \in \{1, \dots, D\}$ fix a spatial step size $\Delta x_d > 0$.

1. The control volumes are the open sets defined by

$$\Omega(i_1, \dots, i_D) := \bigtimes_{d=1}^{D} ((i_d - 1)\Delta x_d, i_d \Delta x_d), \quad (i_1, \dots, i_D) \in \mathbb{Z}^D.$$

2. For each $d \in \{1, \dots, D\}$ define a family of $(D-1)$-dimensional surfaces

$$\partial_d \Omega(i_1, \dots, i_d, \dots, i_D) := [(i_1 - 1)\Delta x_1, i_1 \Delta x_1] \times \cdots \times \{i_d \Delta x_d\} \times \cdots \times [(i_D - 1)\Delta x_D, i_D \Delta x_D].$$

3. The *Finite Volume Parameter set* is the set $\mathcal{D} := \{\Delta t, \Delta x_1, \dots, \Delta x_D\}$.

Remark 4.1.2
We will only ever need to compare parallel boundaries of control volumes, i.e. , the sets $\partial_d \Omega$ will only differ in the dth dimension. So, when it is clear what the other indices are, we adopt the somewhat sloppy notation

$$\partial_d \Omega(i_1, \dots, i_d, \dots, i_D) = \partial_d \Omega(i_d).$$

This notation is adopted with any quantity that depends on the indices $i_1, i_2, \dots, i_D$.

Proposition 4.1.3
For any finite volume parameter set $\mathcal{D} \subset \mathbb{R}^+$ and any $(i_1, \dots, i_D) \in \mathbb{Z}^D$ the boundary of $\Omega(i_1, \dots, i_D)$ is given by

$$\partial\Omega(i_1, \dots, i_D) = \bigcup_{d=1}^{D} \left[\mathrm{cl}\big(\partial_d\Omega(i_d)\big) \cup \mathrm{cl}\big(\partial_d\Omega(i_d - 1)\big)\right].$$

Proposition 4.1.3 follows directly from Definition 4.1.1.2 and will prove very useful when integrating the convection-diffusion equation over control volumes. Since we are dealing with rectangular control volume computing the volumes and surface areas is easy.

Proposition 4.1.4
For any finite volume parameter set $\mathcal{D} \subset \mathbb{R}^+$ and any $(i_1, \dots, i_D) \in \mathbb{Z}^D$ we have

$$\Delta\mathcal{D} := \Big|\big(\Omega(i_1, \dots, i_D)\big)\Big| = \prod_{d=1}^{D} \Delta x_d, \tag{4.3}$$

where $\Big|\Omega(i_1, \dots, i_D)\Big|$ is the D-dimensional Lebesgue measure of $\Omega(i_1, \dots, i_D)$. Furthermore, for any $d \in \{1, \dots, D\}$ we have

$$\Big|\big(\partial_d\Omega(i_1, \dots, i_D)\big)\Big| = \prod_{\substack{k=1 \\ k\neq d}}^{D} \Delta x_k, \tag{4.4}$$

where $\Big|\partial_d\Omega(i_1, \dots, i_D)\Big|$ is the $(D-1)$-dimensional Lebesgue measure of $\partial_d\Omega(i_d)$.

Proposition 4.1.4 follows directly from the definition of the control volumes, the boundaries and the Lebesgue measure in multiple dimensions. Before integrating over the control volumes, the outward pointing unit normal vectors need to be defined.

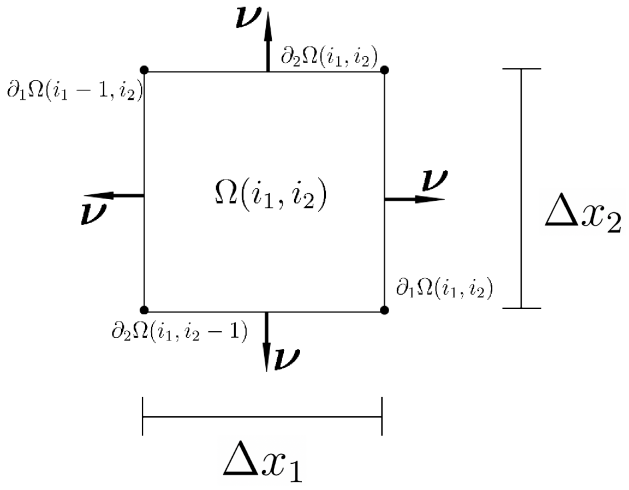
Proposition 4.1.5
For any finite volume parameter set $\mathcal{D} \subset \mathbb{R}^+$ and any $i_1, \dots, i_D \in \mathbb{Z}$ the outward pointing unit normal vector $\boldsymbol{\nu}_{i_1,\dots,i_D}(\boldsymbol{x})$ of the boundary $\partial\Omega(i_1, \dots, i_D)$ is given by

$$\boldsymbol{\nu}_{i_1,\dots,i_D}(\boldsymbol{x}) = \begin{cases} (\delta_{1,d}, \dots, \delta_{D,d}) & \text{if } \boldsymbol{x} \in \partial_d\Omega(i_d), \\ -(\delta_{1,d}, \dots, \delta_{D,d}) & \text{if } \boldsymbol{x} \in \partial_d\Omega(i_d - 1), \end{cases} \tag{4.5}$$

where $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ if $i \neq j$.

Proposition 4.1.5 follows from viewing the surfaces $\partial_d\Omega$ as subsets of hyperplanes in $\mathbb{R}^D$. From this proposition we can conclude that for the forward faces $\partial_d\Omega(i_d)$ the outward pointing unit normal vector is the unit vector along the positive x_d-axis. For the backwards faces $\partial_d\Omega(i_d - 1)$ the outward pointing unit normal vector is the unit vector along the negative x_d-axis. Figure 4.1 illustrates this idea.

Figure 4.1: Boundaries and normal vectors of a 2D control volume.



We now have everything that is needed to start integrating the convection-diffusion equation over the control volumes.

Proposition 4.1.6
For any finite volume parameter set $\mathcal{D} \subset \mathbb{R}^+$ and any $(i_1, \ldots, i_D) \in \mathbb{Z}^D$ define

$$\overline{\phi}(t, i_1, \ldots, i_D) := \Delta\mathcal{D}^{-1} \int_{\Omega(i_1,\ldots,i_D)} \phi(t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}), \tag{4.6}$$

and

$$F_d(t, i_1, \ldots, i_D) := \int_{\partial_d\Omega(i_d)} w_d(\boldsymbol{x}) \frac{\partial\phi}{\partial x_d}(t, \boldsymbol{x}) - v_d(\boldsymbol{x})\phi(t, \boldsymbol{x}) \, \mathrm{d}S(\boldsymbol{x}). \tag{4.7}$$

If ϕ is a sufficiently smooth solution to the initial value problem (4.2) then

$$\frac{\mathrm{d}\overline{\phi}}{\mathrm{d}t}(t, i_1, \ldots, i_D) = \Delta\mathcal{D}^{-1} \sum_{d=1}^{D} \Big[F_d(t, i_d) - F_d(t, i_d - 1) \Big]. \tag{4.8}$$

Proof:
Fix a finite volume parameter set $\mathcal{D} = \{\Delta t, \Delta x_1, \ldots, \Delta x_D\}$ and $i_1, \ldots, i_D \in \mathbb{Z}$. First note that the total flux vector of the convection-diffusion equation is

$$\boldsymbol{f}(\phi, t, \boldsymbol{x}) := -\boldsymbol{w}(\boldsymbol{x}) * \nabla\phi(t, \boldsymbol{x}) + \phi(t, \boldsymbol{x})\boldsymbol{v}(\boldsymbol{x}).$$

The entries of $\boldsymbol{f}$ denoted by $f_1, \ldots, f_D$ are given by

$$f_d(\phi, t, \boldsymbol{x}) = -w_d(\boldsymbol{x}) \frac{\partial\phi}{\partial x_d}(t, \boldsymbol{x}) + \phi(t, \boldsymbol{x}) v_d(\boldsymbol{x}), \ \forall d \in \{1, \ldots, D\}. \tag{4.9}$$

Substituting $\boldsymbol{f}$ into the convection-diffusion equation and integrating over $\Omega(i_1, \ldots, i_D)$ we find

$$\underbrace{\int_{\Omega(i_1,\ldots,i_D)} \frac{\partial\phi}{\partial t}(t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x})}_{=:J_1} = \underbrace{- \int_{\Omega(i_1,\ldots,i_D)} \nabla \cdot \boldsymbol{f}(\phi, t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x})}_{=:J_2}. \tag{4.10}$$

Recall that if ϕ is integrable over $\Omega(i_1,\dots,i_D)$ and its time derivative is also absolutely integrable over $\Omega(i_1,\dots,i_D)$, then by the dominated convergence theorem we can take the derivative outside of the integral in J_1. From (4.6) it then follows that

$$J_1 = \frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega(i_1,\dots,i_D)} \phi(t,\boldsymbol{x})\,\mathrm{d}V(\boldsymbol{x}) = \Delta\mathcal{D}\frac{\mathrm{d}\overline{\phi}}{\mathrm{d}t}(t,i_1,\dots,i_D). \tag{4.11}$$

By applying the divergence theorem and then invoking Proposition 4.1.5 we find

$$\begin{aligned} J_2 &= \int_{\partial\Omega(i_1,\dots)} \boldsymbol{f}(\phi,t,\boldsymbol{x})\cdot\boldsymbol{\nu}_{i_1,\dots,i_D}(\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}) \\ &= \sum_{d=1}^{D}\left[\int_{\partial_d\Omega(i_d)} f_d(\phi,t,\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}) - \int_{\partial_d\Omega(i_d-1)} f_d(\phi,t,\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x})\right] \\ &= \sum_{d=1}^{D}\Big[-F_d(t,i_d)+F_d(t,i_d-1)\Big]. \end{aligned} \tag{4.12}$$

The last line follows directly from (4.7) and (4.9). Combining(4.12) and (4.11) in (4.10) gives the result.

□

Next, to form a discrete approximation of $\overline{\phi}$ the derivatives are discretised. Define

$$\overline{\phi}^n(i_1,\dots,i_D) := \overline{\phi}(n\Delta t, i_1,\dots,i_D),\ \forall(\ i_1,\dots,i_D)\in\mathbb{Z}^D,$$

then approximate the time derivative of $\overline{\phi}$ by taking a forward finite difference;

$$\frac{\mathrm{d}\overline{\phi}}{\mathrm{d}t}(n\Delta t,i_1,\dots,i_D) \approx \frac{\overline{\phi}^{n+1}(i_1,\dots,i_D)-\overline{\phi}^{n}(i_1,\dots,i_D)}{\Delta t}. \tag{4.13}$$

Recall from Section 2.2.1 that the time discretisation can also be done under the integral in J_1. Doing so will lead to the same finite volume method.

Discretising the flux integrals F_d is a little less simple, we break it down into two terms that are discretised in different ways.

$$F_d(i_1,\dots,i_D) = \underbrace{\int_{\partial_d\Omega(i_d)} w_d(\boldsymbol{x})\frac{\partial\phi}{\partial x_d}(t,\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x})}_{=:J_3} - \underbrace{\int_{\partial_d\Omega(i_d)} v_d(\boldsymbol{x})\phi(t,\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x})}_{=:J_4}. \tag{4.14}$$

Assume that for all $t>0$ and each $d\in\{1,\dots,D\}$ the partial derivative $\frac{\partial\phi}{\partial x_d}$ is constant on the boundary $\partial_d\Omega(i_1,\dots,i_D)$ for all $(i_1,\dots,I_D)\in\mathbb{Z}^D$. Also assume that ϕ attains its average value at the center of the control volume $\Omega(i_1,\dots,i_D)$. Using a centred finite difference yields

$$J_3 \approx \frac{\overline{\phi}(t,i_d+1)-\overline{\phi}(t,i_d)}{\Delta x_d}\int_{\partial_d\Omega(i_d)} w_d(\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}) = \frac{\overline{\phi}(t,i_d+1)-\overline{\phi}(t,i_d)}{\Delta x_d}W_d(i_d), \tag{4.15}$$

where we have defined

$$W_d(i_1,\dots,i_D) := \int_{\partial_d\Omega(i_1,\dots,i_D)} w_d(\boldsymbol{x})\,\mathrm{d}S(\boldsymbol{x}),\ \forall\ i_1,\dots,i_D\in\mathbb{Z}. \tag{4.16}$$

This is a generalised version of the discretisation of the heat equation in Section 2.3. To discretise the integral J_4 we use the upwinding scheme used in [40] and Example 2.2.4.

$$J_4 \approx \min\{0, V_d(i_d)\}\overline{\phi}(t, i_d + 1) + \max\{0, V_d(i_d)\}\overline{\phi}(t, i_d), \tag{4.17}$$

where we defined

$$V_d(i_1, \ldots, i_D) := \int_{\partial_d \Omega(i_1,\ldots,i_D)} v_d(\boldsymbol{x}) \, \mathrm{d}S(\boldsymbol{x}), \ \forall \ i_1, \ldots, i_D \in \mathbb{Z}. \tag{4.18}$$

This approximation takes the viewpoint that when the convection flux across the boundary (the flux of convection velocity field) is negative, i.e., inward, then density flows from the neighbouring control volume into the current control volume. On the other hand, if the convection flux is positive, i.e., outward, then density flows from the current control volume into the neighbouring control volume.

Combining (4.15) and (4.17) gives the approximation

$$F_d(t, i_1, \ldots, i_D) = \overline{\phi}(t, i_d + 1) \left(\frac{W_d(i_d)}{\Delta x_d} - \min\{0, V_d(i_d)\} \right) + \overline{\phi}(t, i_d) \left(-\frac{W_d(i_d)}{\Delta x_d} - \max\{0, V_d(i_d)\} \right). \tag{4.19}$$

It then follows that for all $(i_1, \ldots, i_D) \in \mathbb{Z}^D$ and $t > 0$ we have that the total flux through the boundaries of the control volume $\Omega(i_1, \ldots, i_D)$ along the x_d-axis is given by

$$\begin{aligned} F_d(t, i_d) - F_d(t, i_d - 1) &= \overline{\phi}(t, i_d + 1) \left(\frac{W_d(i_d)}{\Delta x_d} - \min\{0, V_d(i_d)\} \right) \\ &\quad + \overline{\phi}(t, i_d) \left(-\frac{W_d(i_d) + W_d(i_d - 1)}{\Delta x_d} - \max\{0, V_d(i_d)\} + \min\{0, V_d(i_d - 1)\} \right) \\ &\quad + \overline{\phi}(t, i_d - 1) \left(\frac{W_d(i_d - 1)}{\Delta x_d} + \max\{0, V_d(i_d - 1)\} \right) \\ &= \overline{\phi}(t, i_d + 1) U_d(i_d + 1) + \overline{\phi}(t, i_d) C_d(i_d) + \overline{\phi}(t, i_d - 1) L_d(i_d - 1), \end{aligned} \tag{4.20}$$

where

$$U_d(i_1, \ldots, i_D) := \frac{W_d(i_d - 1)}{\Delta x_d} - \min\{0, V_d(i_d - 1)\} \tag{4.21}$$

$$C_d(i_1, \ldots, i_D) := -\frac{W_d(i_d) + W_d(i_d - 1)}{\Delta x_d} - \max\{0, V_d(i_d)\} + \min\{0, V_d(i_d - 1)\} \tag{4.22}$$

$$L_d(i_1, \ldots, i_D) := \frac{W_d(i_d)}{\Delta x_d} + \max\{0, V_d(i_d)\}. \tag{4.23}$$

Putting (4.8),(4.13) and (4.20) together we find for $t = n\Delta t$

$$\begin{aligned} \overline{\phi}^{n+1}(i_1, \ldots, i_D) &= \Delta t \Delta \mathcal{D}^{-1} \sum_{d=1}^{D} U_d(i_d + 1) \overline{\phi}^n(i_d + 1) + \Delta t \Delta \mathcal{D}^{-1} \sum_{d=1}^{D} L_d(i_d - 1) \overline{\phi}^n(i_d - 1) \\ &\quad + \left(1 + \Delta t \Delta \mathcal{D}^{-1} \sum_{d=1}^{D} C_d(i_d) \right) \overline{\phi}^n(i_1, \ldots, i_D). \end{aligned} \tag{4.24}$$

With (4.24) the discrete approximation of ϕ can be defined.

Definition 4.1.7
Fix a finite volume parameter set $\mathcal{D} \subset \mathbb{R}^+$, let

$$\overline{\phi}^0(i_1, \ldots, i_D) = \Delta\mathcal{D}^{-1} \int_{\Omega(i_1,\ldots,i_D)} \phi_0(\boldsymbol{x}) \, \mathrm{d}V\boldsymbol{x}, \ \forall i_1, \ldots, i_D \in \mathbb{Z}.$$

The approximate solution $\phi_\mathcal{D} : [0, \infty) \times \mathbb{R}^D \to \mathbb{R}$ to the initial value problem (4.2) is defined as the piecewise constant function satisfying

$$\phi_\mathcal{D}(t, \boldsymbol{x}) = \overline{\phi}^n(i_1, \ldots, i_D), \ \forall \ (t, \boldsymbol{x}) \in [n\Delta t, (n+1)\Delta t) \times \Omega(i_1, \ldots, i_D), \tag{4.25}$$

where $\overline{\phi}^n(i_1, \ldots, i_D)$ satisfies (4.24) for all $n \in \mathbb{N}$ and $(i_1, \ldots, i_D) \in Z^D$.

At the start of this chapter, it was claimed that the finite volume method presented here will be positivity and integral preserving. It has not yet been ensured that positivity will be preserved, but it has been ensured that the integral is preserved.

Lemma 4.1.8
Fix a finite volume parameter set $\mathcal{D} \subset \mathbb{R}^+$ and let $\phi_\mathcal{D}$ be the approximate solution from Definition 4.1.7. For any $t \geq 0$,

$$\int_{\mathbb{R}^D} \phi_\mathcal{D}(t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) = \int_{R_d} \phi_0(\boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}).$$

Proof:
Notice that it follows directly from (4.21),(4.22) and (4.23) that U_d, C_d and L_d add to zero, that is for all $(i_1, \ldots, i_D) \in \mathbb{Z}^D$

$$U_d(i_1, \ldots, i_D) + C_d(i_1, \ldots, i_D) + L_d(i_1, \ldots, i_D) = 0. \tag{4.26}$$

For any $n \in \mathbb{N}$ it then follows that

$$\begin{aligned}
&\Delta\mathcal{D} \sum_{(i_1,\ldots,i_D)\in\mathbb{Z}^D} \overline{\phi}^{n+1}(i_1, \ldots, i_D) \\
=&\Delta\mathcal{D} \sum_{(i_1,\ldots,i_D)\in\mathbb{Z}^D} \overline{\phi}^{n}(i_1, \ldots, i_D) \\
&+ \Delta t \sum_{(i_1,\ldots,i_D)\in\mathbb{Z}^D} \sum_{d=1}^{D} \Big[U_d(i_d+1)\overline{\phi}^n(i_d+1) + C_d(i_d)\overline{\phi}^n(i_d) + L_d(i_d-1)\overline{\phi}^n(i_d-1) \Big] \\
=&\Delta\mathcal{D} \sum_{(i_1,\ldots,i_D)\in\mathbb{Z}^D} \overline{\phi}^{n}(i_1, \ldots, i_D) + \Delta t \sum_{(i_1,\ldots,i_D)\in\mathbb{Z}^D} \overline{\phi}^n(i_1, \ldots, i_D) \sum_{d=1}^{D} \Big[U_d(i_d) + C_d(i_d) + L_d(i_d) \Big] \\
=&\Delta\mathcal{D} \sum_{(i_1,\ldots,i_D)\in\mathbb{Z}^D} \overline{\phi}^{n}(i_1, \ldots, i_D).
\end{aligned}$$

By the definition of $\phi_\mathcal{D}$ we know that;

$$\int_{\mathbb{R}^D} \phi_\mathcal{D}(t, \boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}) = \Delta\mathcal{D} \sum_{(i_1,\ldots,i_D)\in\mathbb{Z}^D} \overline{\phi}^{n}(i_1, \ldots, i_D),$$

and the result then follows by induction on n. □

Lemma 4.1.8 shows that the integral is preserved with any finite volume parameter set. However, to ensure that positivity is preserved the set of time discretisation steps that are allowed has to be restricted.

Lemma 4.1.9
For any finite volume parameter set $\mathcal{D} \subset \mathbb{R}^+$, define

$$C(i_1, \ldots, i_D) := \sum_{d=1}^{D} C_d(i_1, \ldots, i_D), \ \forall(\ i_1, \ldots, i_D) \in \mathbb{Z}^D.$$

If there exists $\varepsilon \in [0, 1)$ such that

$$\Delta t \leq \Delta\mathcal{D}(1-\varepsilon)\left(\max_{(i_1,\ldots,i_D)\in\mathbb{Z}^D}\{-C(i_1, \ldots, I_D)\}\right)^{-1} \tag{4.27}$$

then

$$\phi_{\mathcal{D}}(t, \boldsymbol{x}) \geq 0, \ \forall \boldsymbol{x} \in \mathbb{R}^D, \ t > 0.$$

Proof:
By the definition of $\overline{\phi}^0$, if $\phi_0(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^D$, then $\overline{\phi}^0(i_1, \ldots, i_D) \geq 0$ for all $(i_1, \ldots, i_D) \in \mathbb{Z}^D$. Recall that $\boldsymbol{w}$ is a non-negative vector field, therefore all W_d are also non-negative. This implies that U_d and L_d are also non-negative, and C_d is non-positive. Now suppose that for $n \in \mathbb{N}$ and $\varepsilon \in [0, 1)$ the time step Δt satisfies (4.27) and $\overline{\phi}^n(i_1, \ldots, i_D) \geq 0$ for all $(i_1, \ldots, i_D) \in \mathbb{Z}^D$. The finite volume recurrence (4.24) then implies that

$$\overline{\phi}^{n+1}(i_1, \ldots, i_D) \geq \left(1 + \Delta t \Delta\mathcal{D}^{-1} \sum_{d=1}^{D} C_d(i_d)\right) \overline{\phi}^n(i_1, \ldots, i_D) \geq 1 - (1-\varepsilon) \geq 0.$$

The result then follows by induction on n.

□

After deriving a finite volume method and showing that it preserves integrals and positivity Fox et al. also state an error estimate for their finite volume scheme. This estimate follows from Theorem 2 in [38] , the proof of which relies heavily on the existence of locally integrable weak solutions to the continuity equation (functions that solve the integral form of the continuity equation). Such solutions are shown to exist in Theorem 1 of [38]. However, we do not have such general existence theorems for convection-diffusion equations. Furthermore, the proof of Theorem 2 in [38] does not necessarily extend to convection-diffusion equations. Convergence of a similar finite volume method on a bounded domain with Dirichlet boundary conditions is treated in [9, Sec. 17.1]. We instead resort to case-by-case numerical estimates of the error when we implement our finite volume method.

4.2 2D Example

To check that our new finite volume method works we tested it on a two-dimensional example. Let $k > 0$, and consider the initial value problem

$$\begin{cases} \frac{\partial \phi}{\partial t} - k\frac{\partial^2 \phi}{\partial x_1^2} - k\frac{\partial^2 \phi}{\partial x_2^2} + x_2\frac{\partial \phi}{\partial x_1} - x_1\frac{\partial \phi}{\partial x_2} = 0, & \forall t > 0,\ \boldsymbol{x} \in \mathbb{R}^2, \\ \phi(0, \boldsymbol{x}) = \phi_0(\boldsymbol{x}), & \forall \boldsymbol{x} \in \mathbb{R}^2. \end{cases} \tag{4.28}$$

It can be shown by the method of characteristics that the solution of this initial value problem is a rotating solution of the heat equation. In particular

$$\phi(t, \boldsymbol{x}) = \widehat{\phi}(t, \boldsymbol{R}_t\boldsymbol{x}) \ \forall\ t \geq 0,\ \boldsymbol{x} \in \mathbb{R}^2, \tag{4.29}$$

where $\boldsymbol{R}_t$ is a matrix representing an anti-clockwise rotation of t radians about the origin, $\boldsymbol{x}$ is viewed as a column vector and $\widehat{\phi}$ is a solution to following initial value problem;

$$\begin{cases} \frac{\partial \widehat{\phi}}{\partial t} - k\frac{\partial^2 \widehat{\phi}}{\partial x_1^2} - k\frac{\partial^2 \widehat{\phi}}{\partial x_2^2} = 0, & \forall t > 0,\ \boldsymbol{x} \in \mathbb{R}^2, \\ \widehat{\phi}(0, \boldsymbol{x}) = \phi_0(\boldsymbol{x}), & \forall \boldsymbol{x} \in \mathbb{R}^2. \end{cases} \tag{4.30}$$

We approximate the solution in a bounded region. For this reason, we need to make sure that the exact solution is well localised in a bounded region. We did this by choosing a version of the fundamental solution to the heat equation. Let ψ be the two-dimensional fundamental solution of the heat equation, then

$$\psi(t, \boldsymbol{x}) = \frac{1}{4k\pi t} \exp\left(-\frac{x_1^2 + x_2^2}{4kt}\right)$$

At $t = 0$ this function is equal to a Dirac delta distribution. However, we didn't take the Dirac delta distribution as our initial condition as this requires unnecessary work to determine which control volume contains the singularity of the distribution. Instead, a shifted fundamental solution was chosen. Let

$$\phi_0(\boldsymbol{x}) = \psi(t_1, \boldsymbol{x} - \boldsymbol{x}_0), \tag{4.31}$$

for some $t_1 > 0$ and $\boldsymbol{x}_0 \in \mathbb{R}^2$. It then follows that

$$\widehat{\phi}(t, \boldsymbol{x}) = \psi(t + t_1, \boldsymbol{x} - \boldsymbol{x}_0).$$

The fact that the heat equation is rotation and translation invariant along with (4.29) implies that

$$\phi(t, \boldsymbol{x}) = \psi(t + t_1, \boldsymbol{R}_t\boldsymbol{x} - \boldsymbol{x}_0).$$

We will approximate this solution on $(-1, 1)^2$ at $t = T$, for some $T > 0$. The control volumes are then given by

$$\Omega(i_1, i_2) = (-1 + (i_1 - 1)\Delta x_1, -1 + i_1\Delta x_1) \times (-1 + (i_2 - 1)\Delta x_2, -1 + i_2\Delta x_2),$$

where

$$\Delta x_1 := \frac{2}{I_1} \quad \text{and} \quad \Delta x_2 := \frac{2}{I_2},$$

for some positive integers I_1, I_2. We adjust the definition of the boundaries $\partial_1\Omega$ and $\partial_2\Omega$ in a similar way (see Section 2.3). The index i_1 now runs from 1 to I_1, and similarly the

index i_2 runs from 1 to I_2.

The fact that we are considering a bounded domain means that boundary conditions are needed. A simple zero-flux boundary condition was chosen. In this case the matrices U_d, C_d and L_d have to be adjusted so that the boundary terms are zero. We do this by creating auxiliary coefficient matrices;

$$\widehat{U}_d(i_1,i_2) := \begin{cases} 0 & \text{if } i_d = 1, \\ \frac{W_d(i_d-1)}{\Delta x_d} - \min\{0, V_d(i_d-1)\} & \text{otherwise.} \end{cases} \tag{4.32}$$

$$\widehat{L}_d(i_1,i_2) := \begin{cases} 0 & \text{if } i_d = I_d, \\ \frac{W_d(i_d)}{\Delta x_d} + \max\{0, V_d(i_d)\} & \text{otherwise.} \end{cases} \tag{4.33}$$

and

$$C_d(i_1,i_2) := \begin{cases} -\frac{W_d(i_d)}{\Delta x_d} - \max\{0, V_d(i_d)\} & \text{if } i_d = 1, \\ -\frac{W_d(i_d)+W_d(i_d-1)}{\Delta x_d} - \max\{0, V_d(i_d)\} + \min\{0, V_d(i_d-1)\} & \text{if } 1 < i_d < I_d, \\ -\frac{W_d(i_d-1)}{\Delta x_d} + \min\{0, V_d(i_d-1)\} & \text{if } i_d = I_d. \end{cases} \tag{4.34}$$

Then the coefficient matrices for the finite volume method are given by

$$U_d = \texttt{circshift}(\widehat{U}_d, -1, d),\ L_d = \texttt{circshift}(\widehat{L}_d, 1, d) \quad \text{and} \quad C = C_1 + C_2. \tag{4.35}$$

for both $d = 1$ and $d = 2$. The function `circhsift` is the Matlab circular shifting operation that shifts the dth index by the number of spaces specified by the second input. The shifting is circular because entries that move beyond the limits of the index are shifted around to the other end of the index. For an interior control volume ($1 < i_d < I_D$ for $d = 1$ and $d = 2$) we have that

$$\begin{aligned} \overline{\phi}^{n+1}(i_1,i_2) = &\left[\Delta t(\Delta x_1\Delta x_2)^{-1}U_2(i_1,i_2)\right]\overline{\phi}^n(i_1,i_2+1) + \left[\Delta t(\Delta x_1\Delta x_2)^{-1}U_1(i_1,i_2)\right]\overline{\phi}^n(i_1+1,i_2) \\ &+ \left[1+\Delta t(\Delta x_1\Delta x_2)^{-1}C(i_1,i_2)\right]\overline{\phi}^n(i_1,i_2) \\ &+ \left[\Delta t(\Delta x_1\Delta x_2)^{-1}L_1(i_1,i_2)\right]\overline{\phi}^n(i_1-1,i_2) + \left[\Delta t(\Delta x_1\Delta x_2)^{-1}L_2(i_1,i_2)\right]\overline{\phi}^n(i_1,i_2-1). \end{aligned}$$

If $i_i = 1$ the $\overline{\phi}^n(i_1-1,i_2)$ term refers to a control volume outside of $(-1,1)^2$. The zero flux boundary condition means the flux across the boundaries of $(-1,1)^2$ is zero. So, we drop the terms that refer to such control volumes

$$\begin{aligned} \overline{\phi}^{n+1}(1,i_2) = &\left[\Delta t(\Delta x_1\Delta x_2)^{-1}U_2(1,i_2)\right]\overline{\phi}^n(1,i_2+1) + \left[\Delta t(\Delta x_1\Delta x_2)^{-1}U_1(1,i_2)\right]\overline{\phi}^n(2,i_2) \\ &+ \left[1+\Delta t(\Delta x_1\Delta x_2)^{-1}C(1,i_2)\right]\overline{\phi}^n(1,i_2) \\ &+ \left[\Delta t(\Delta x_1\Delta x_2)^{-1}L_2(1,i_2)\right]\overline{\phi}^n(1,i_2-1). \end{aligned}$$

Similarly, if $i_1 = I_1$ we have

$$\begin{aligned} \overline{\phi}^{n+1}(I_1,i_2) = &\left[\Delta t(\Delta x_1\Delta x_2)^{-1}U_2(I_1,i_2)\right]\overline{\phi}^n(I_1,i_2+1) \\ &+ \left[1+\Delta t(\Delta x_1\Delta x_2)^{-1}C(I_1,i_2)\right]\overline{\phi}^n(I_1,i_2) \\ &+ \left[\Delta t(\Delta x_1\Delta x_2)^{-1}L_1(I_1,i_2)\right]\overline{\phi}^n(I_1-1,i_2) + \left[\Delta t(\Delta x_1\Delta x_2)^{-1}L_2(I_1,i_2)\right]\overline{\phi}^n(I_1,i_2-1). \end{aligned}$$

The second index i_2 is dealt with in the same way. We already removed references to control volumes outside of $(-1,1)^2$ in the definition of C, which is why we left the C term unchanged (see Section 2.2.2 for the treatment of zero-flux boundary conditions and Section 2.3 for an example). These coefficients still add to zero and the proof of Lemma 4.1.8 can be repeated to show that that the integral will be preserved. It is possible to simplify these matrices using the fact that the diffusion coefficients are constant in space and time. However, to keep the finite volume method in this example general we still view the diffusion coefficients as functions in space.

The time step Δt was chosen to be the largest time step that preserves positivity and exactly divides $[0,T]$ into an integer number of time intervals. So, we took

$$\Delta t = T\left\lceil T\Delta x_1 \Delta x_2 \max_{i_1,i_2}\{-C(i_1,i_2)\}\right\rceil^{-1}, \tag{4.36}$$

where $\lceil\cdot\rceil$ represents the ceiling function, i.e. the smallest integer larger than the argument. The number of time intervals or time steps, Nt is then given by

$$N_t := T\Delta t^{-1}. \tag{4.37}$$

The zero flux boundary conditions will be accurate if the solution is close to zero at the boundaries. If t_1 is small, the initial condition is well localised around $\boldsymbol{x}_0$. For this reason, we chose $\boldsymbol{x_0} = (0.2, 0)$ and $t_1 = 0.05$. We chose $T = \pi/2$, as this represents a quarter rotation around the origin it gives us some more visual intuition about the solution. To prevent the solution from spreading out too much due to diffusion we chose a small diffusion coefficient $k = 0.01$. The integral functions W_d and V_d are evaluated with a mid point rule on the boundaries of the control volumes. The initial condition was approximated with a midpoint rule inside the control volumes, which means that the approximate initial condition is given by the matrix

$$\overline{\boldsymbol{\Phi}}^0(i_1,i_2) = \phi_0\left(-1+\left(i_1-\frac{1}{2}\right)\Delta x_1, -1+\left(i_2-\frac{1}{2}\right)\Delta x_2\right).$$

The subsequent approximations are then given by the recurrence (4.24) with the new coefficient matrices given by (4.35). To implement this recurrence we create an $(I_1I_2)\times(I_1I_2)$ diagonal matrix $\boldsymbol{M}$, which contains the entries of $\Delta t U_1$ on the first upper diagonal, the entries of $\Delta t U_2$ on the I_1th upper diagonal. The entries of $\Delta t L_1$ and $\Delta t L_2$ are on the first and I_1th lower diagonals, respectively. The main diagonal of $\boldsymbol{M}$ contains the entries of $\boldsymbol{I} + \Delta t C$. After vectorising $\overline{\boldsymbol{\Phi}}^0$ to give $\boldsymbol{\phi}^0$ we compute the subsequent approximations by

$$\boldsymbol{\phi}^{n+1} = \boldsymbol{M}\boldsymbol{\phi}^n \quad \forall n = 1,\ldots,N_t. \tag{4.38}$$

We call the process of computing successive $\boldsymbol{\phi}^n$'s by this recursion the *Integration step* since it is a discrete analogue of integrating the convection-diffusion equation over the time-interval spanning from the initial time to the final time. We call the process of setting the transition matrix $\boldsymbol{M}$ the *setup*.

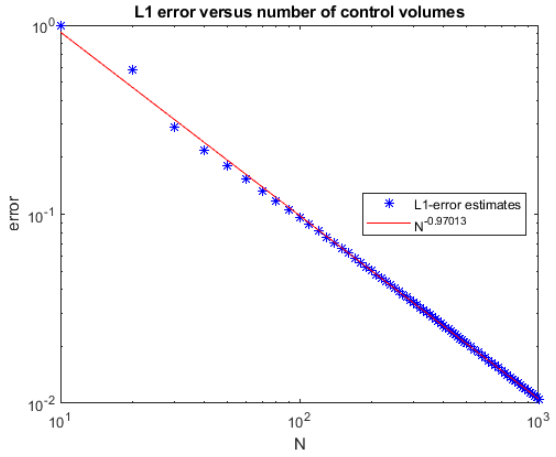
Implementing this method using matrices we can expect the computation time to scale as $\mathcal{O}(I^2N_t)$, at best. We know this because there are I_1I_2 values that make up the approximation of ϕ, and there are N_t approximations to compute. To measure convergence, we used the L^1 norm between the exact solution and the approximation at

time $t = T$, which we computed by a mid-point quadrature rule. This is an estimate of the total error that accumulates at each time step, which is called the *global error*. We ran the finite volume method for $I_1 = I_2 = N$ with N ranging from 20 to 1000 and got the summary plots in Figure 4.2.

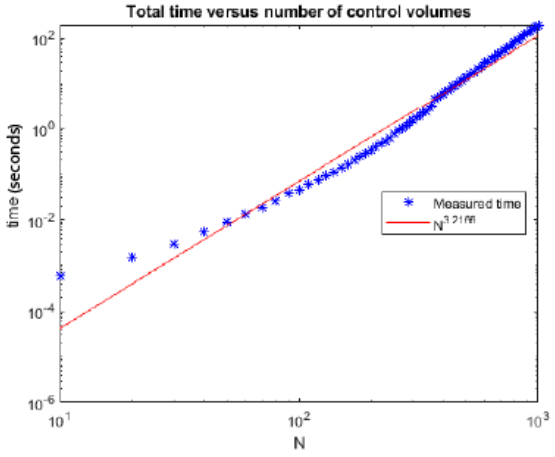
From these plots we have experimental evidence that our numerical method converges for this case. In fact from Figure 4.2a we estimate the numerical order of convergence to be 0.97013, that is we estimate that the global error (the final error at time $t = T$) decreases as $\mathcal{O}(N^{-0.97013})$. We can also see from Figure 4.2b that the total time taken grows as $\mathcal{O}(N^{3.2166})$, and from Figure 4.2d we see that the main reason for growth in computation time is the time taken for the integration step (successive computations of (4.38)). There are two reasons why this is the case. First, the number of entries that need to be computed is N^2 and we compute these entries by a product of an $N^2 \times N^2$ matrix and a $N^2 \times 1$ vector. This requires requires $\mathcal{O}(N^2)$ operations, so as N grows it takes longer to perform one time step. The second reason for the growth in computation time is that the number of time steps, N_t, is estimated to grow as $\mathcal{O}(N^{1.6303})$. So, more and more matrix-vector products need to be computed as N grows, which increases the total time spent in the integration step.

In general, the total computation time of this method will grow exponentially in the dimension since there are $\mathcal{O}(N^D)$ entries that need to be computed. This would limit the usefulness of our method in higher dimensions. However, this finite volume method can be rewritten in terms of tensors and tensor operations which allows us to utilise the tensor train format to reduce the computational cost. This is described in the next chapter.

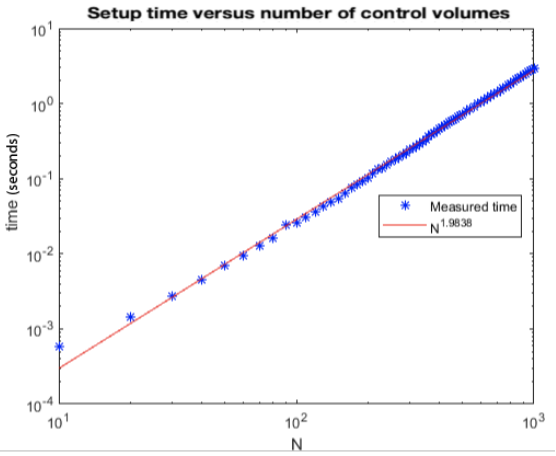
Figure 4.2: Summary Plots



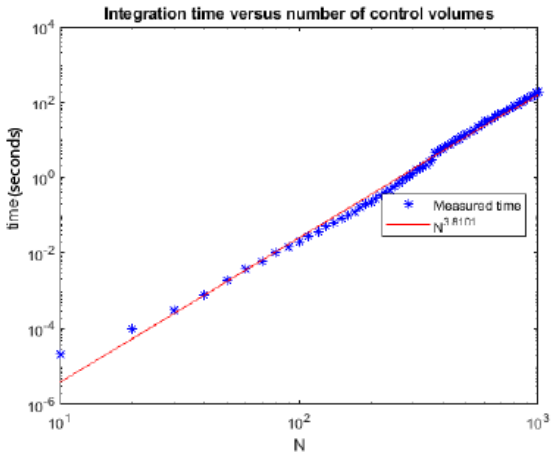
(a) Global L^1 error.



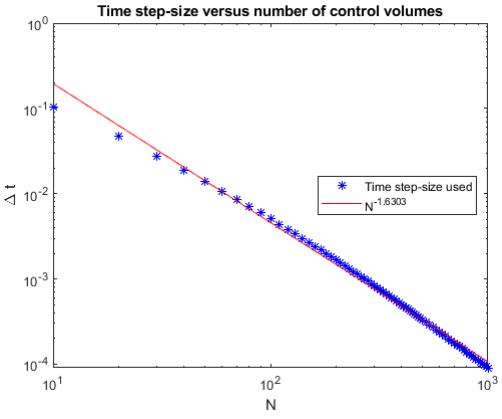
(b) Total Computation time.



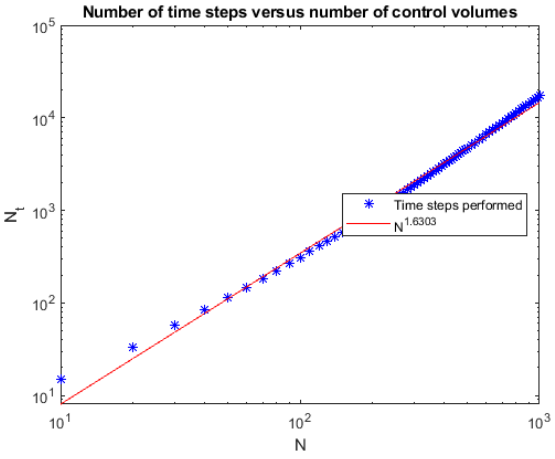
(c) Time taken to set transition matrix.



(d) Integrations step computation time



(e) Growth of Δt.



(f) Growth of N_t.

Chapter 5

Finite Volume Method as a Tensor Train Recursion

In the previous chapter we saw that the usual matrix-vector method for performing the integration step in our finite volume method leads to a computational cost that scales exponentially in dimension. Tensor train decompositions allow us to get around this issue by performing all the basic operations in the TT format. In this chapter we outline the implementation of the finite volume method from the previous chapter in TT format.

Recall from (4.24) that our finite volume method is given by the recursion

$$\begin{aligned}\overline{\phi}^{n+1}(i_1,\dots,i_D) = \Delta t\Delta\mathcal{D}^{-1}\sum_{d=1}^{D} U_d(i_d+1)\overline{\phi}^{n}(i_d+1) \\ + \left(1+\Delta t\Delta\mathcal{D}^{-1}\sum_{d=1}^{D} C_d(i_d)\right)\overline{\phi}^{n}(i_1,\dots,i_D) \\ + \Delta t\Delta\mathcal{D}^{-1}\sum_{d=1}^{D} L_d(i_d-1)\overline{\phi}^{n}(i_d-1),\end{aligned}$$

where the U_d, C_d and L_d are given by (4.21),(4.22) and (4.23). In Chapter 4 we ignored the fact that these coefficients and $\overline{\phi}^n$ are Dth-order tensors, so as not to distract from the finite volume method that we were developing. However, we will now take advantage of this fact. To indicate that U_d, C_d, L_d and $\overline{\phi}^n$ are now viewed as tensors we adopt the notation;

$$\overline{\boldsymbol{\Phi}}^n := \overline{\phi}^n \quad \forall\ n \in \mathbb{N},$$

as well as

$$\boldsymbol{\mathcal{U}}_d := U_d,\ \boldsymbol{\mathcal{C}}_d := C_d, \quad \text{and} \quad \boldsymbol{\mathcal{L}}_d := L_d \quad \forall\ d \in \{1,\dots,D\}.$$

As in Lemma 4.1.9 take $\boldsymbol{\mathcal{C}}$ to be the sum of all $\boldsymbol{\mathcal{C}}_d$ and define

$$\boldsymbol{\mathcal{M}} := 1 + \Delta t\Delta\mathcal{D}^{-1}\boldsymbol{\mathcal{C}},$$

where $\Delta\mathcal{D}$ is defined as the product of the spatial grid spacings as in Proposition 4.1.4 and Δt is a positivity preserving time step-size.

Rewrite (4.24) as

$$\begin{aligned}\overline{\boldsymbol{\Phi}}^{n+1}(i_1,\ldots,i_D) = \Delta t\Delta\mathcal{D}^{-1}\sum_{d=1}^{D}\boldsymbol{\mathcal{U}}_d(i_d+1)\overline{\boldsymbol{\Phi}}^{n}(i_d+1)\\ + \boldsymbol{\mathcal{M}}(i_1,\ldots,i_d)\overline{\boldsymbol{\Phi}}^{n}(i_1,\ldots,i_D)\\ + \Delta t\Delta\mathcal{D}^{-1}\sum_{d=1}^{D}\boldsymbol{\mathcal{L}}_d(i_d-1)\overline{\boldsymbol{\Phi}}^{n}(i_d-1).\end{aligned} \tag{5.1}$$

After defining certain index shifting operations for tensors (5.1) may be stated as the sum of Hadamard products.

Definition 5.0.1

1. Let $\boldsymbol{\mathcal{X}}$ be an infinite Nth-order tensor. For any $n \in \{1,\ldots,N\}$ and $k \in \mathbb{Z}$ define the *n-mode index shift operator* $\tau_{n,k}$ by

$$\tau_{n,k}[\boldsymbol{\mathcal{X}}](i_1,\ldots,i_n,\ldots,i_N) = \boldsymbol{\mathcal{X}}(i_n-k),\ \forall\ i_1,\ldots,i_N \in \mathbb{Z}, \tag{5.2}$$

   where all i_m with $m \neq n$ are left unchanged. So, $\tau_{n,k}$ shifts the nth index forward by k positions. If k is negative, then the index gets shifted backwards.

2. Let $\boldsymbol{\mathcal{X}}$ be a finite Nth-order tensor with mode sizes $I_1,\ldots,I_N$. For any $n \in \{1,\ldots,N\}$ and any integer $k \in \{-I_n+1,\ldots,-1,1,\ldots,I_n-1\}$ define the *n-mode circular index shift operator* $\tau^{circ}_{n,k}$ by

$$\tau^{circ}_{n,k}[\boldsymbol{\mathcal{X}}](i_1,\ldots,i_n,\ldots,i_N) = \boldsymbol{\mathcal{X}}\left((i_n-k)\text{mod } I_n\right) \tag{5.3}$$

   for all $i_1,\ldots,i_N \in \mathbb{Z}$. Again, every i_m with $m \neq n$ is left unchanged. This is called the circular shifting operator because it treats $i_n = 1$ and $i_n = I_n$ as adjacent "columns".

The circular shifting operator is a bit hard to understand from the definition, but an example should make it clear.

Example 5.0.2
Let $\boldsymbol{A}$ be defined as

$$\boldsymbol{A} := \begin{bmatrix}1 & 4 & 7\\ 2 & 5 & 8\\ 3 & 6 & 9\end{bmatrix},$$

then we have

$$\tau^{circ}_{1,1}[\boldsymbol{A}] = \begin{bmatrix}3 & 6 & 9\\ 1 & 4 & 7\\ 2 & 5 & 8\end{bmatrix} \quad \text{and} \quad \tau^{circ}_{2,1}[\boldsymbol{A}] = \begin{bmatrix}7 & 1 & 4\\ 8 & 2 & 5\\ 9 & 3 & 6\end{bmatrix},$$

as well as

$$\tau^{circ}_{1,-1}[\boldsymbol{A}] = \begin{bmatrix}2 & 5 & 8\\ 3 & 6 & 9\\ 1 & 4 & 7\end{bmatrix} \quad \text{and} \quad \tau^{circ}_{2,-1}[\boldsymbol{A}] = \begin{bmatrix}4 & 7 & 1\\ 5 & 8 & 2\\ 6 & 9 & 3\end{bmatrix}.$$

Proposition 5.0.3
Let $\mathcal{D} = \{\Delta t, \Delta x_1, \ldots, \Delta x_D\}$ be any finite volume parameter set. Also let the coefficient tensors $\boldsymbol{\mathcal{U}}_d, \boldsymbol{\mathcal{C}}_d$ and $\boldsymbol{\mathcal{L}}_d$ be given by (4.21),(4.22) and (4.23) and

$$\boldsymbol{\mathcal{M}} = \mathbf{1} + \Delta t \Delta \mathcal{D}^{-1} \sum_{d=1}^{D} \boldsymbol{\mathcal{C}}_d,$$

where $\mathbf{1}$ is a tensor with each entry equal to 1. Then given an initial condition $\overline{\boldsymbol{\Phi}}^0$, for every positive integer n we have

$$\overline{\boldsymbol{\Phi}}^{n+1} = \boldsymbol{\mathcal{M}} * \overline{\boldsymbol{\Phi}}^n + \Delta t \Delta \mathcal{D}^{-1} \sum_{d=1}^{D} \left(\tau_{d,-1}[\boldsymbol{\mathcal{U}}_d * \overline{\boldsymbol{\Phi}}^n] + \tau_{d,1}[\boldsymbol{\mathcal{L}}_d * \overline{\boldsymbol{\Phi}}^n] \right). \tag{5.4}$$

Replace the $\tau_{d,\pm 1}$ with the circular shift $\tau^{circ}_{d,\pm 1}$ for each d when working on a bounded domain.

Proposition 5.0.3 follows directly from the index form (5.1) and the definition of the shifting operators. Using the tensor recursion (5.4) one could implement our finite volume method exactly if all the tensors are given in full. However, simply working with full tensors won't significantly reduce the computational cost. Only the need for vectorising $\overline{\boldsymbol{\Phi}}$ is removed. The benefit of (5.4) is that it allows all $\overline{\boldsymbol{\Phi}}^n$ to be computed in TT format if the initial condition $\overline{\boldsymbol{\Phi}}^0$ and all the coefficient tensors are given in TT format. This reduces the computational cost because all algebraic operations can be done in TT format.

5.1 Towards a Practical Algorithm

In the rest of this chapter, we focus on creating a general method to implement to finite volume method from Chapter 4 using tensor train decompositions. Therefore, we have to assume that all of the tensors that we are dealing with have finite mode sizes. Assume that we are approximating a D-dimensional function ϕ on a rectangular bounded domain Ω at time $t = T$ for some $T > 0$. Let ϕ_0 denote the initial value of ϕ. As usual we denote the diffusion coefficient by $\boldsymbol{w}$ and the convection velocity field by $\boldsymbol{v}$. We define the domain Ω as

$$\Omega := \bigtimes_{d=1}^{D} (x_d^{min}, x_d^{max}),$$

where $x_d^{min}, x_d^{max} \in \mathbb{R}$ and $x_d^{min} < x_d^{max}$ for all $d \in \{1, \ldots, D\}$. The spatial step-sizes are defined by

$$\Delta x_d := \frac{x_d^{max} - x_d^{min}}{I_d} \ \forall \ d \in \{1, \ldots, D\}, \tag{5.5}$$

for some positive integers $I_1, \ldots, I_D$. Each index i_d now runs from 1 to I_d, for all d. The time step-size Δt will be chosen to be positivity preserving. Control volumes are defined analogously to the control volumes used in the 2-dimensional example of Section 4.2;

$$\Omega(i_1, \ldots, i_D) := \bigtimes_{d=1}^{D} (x_d^{min} + (i_d - 1)\Delta x_d, x_d^{min} + i_d \Delta x_d), \tag{5.6}$$

Boundary faces of these control volumes are adjusted by shifting the factor intervals by x_d^{min} for each d. That is for each $d \in \{1, \ldots, D\}$

$$\partial_d\Omega(i_1, \ldots, i_D) := \{\boldsymbol{x} \in \Omega \big| x_d = x_d^{min} + i_d \Delta x_d \text{ and } x_k \in [x_k^{min} + (i_k - 1)\Delta x_k, x_k^{min} + i_k \Delta x_k] \, \forall k \neq d\}$$

for all $i_d \in \{0, 1, \ldots, I_d\}$ and $i_k \in \{1, \ldots, I_k\}$ for all $k \neq d$.

Remark 5.1.1
When implementing our algorithms in Matlab for all tensor operations done in the TT format we use the TT-toolbox [45] by Oseledets.

5.1.1 Initial Setup

Generating a discrete initial condition and the coefficient tensors requires the computation of integrals. Unless these integrals are simple to compute analytically quadrature rules will need to be used. The main goal is to maximise computational efficiency while maintaining acceptable accuracy. An easy way to estimate integrals quickly is by the mid-point rule. The mid-point rule assumes that the value of the function at the centre of the domain of integration is a good approximation of the average value in that domain. This is a valid assumption if the function is not highly variable (e.g., a low frequency sinusoidal wave) and the domain of integration is small. Usually, the first of these conditions will hold for the convection-diffusion equations that we will consider. If we make our control volumes small enough the second condition will hold for our finite volume method. If either of these conditions do not hold one would need to consider these integrals more closely.

Recall from Definition 4.1.7 that given an initial value function ϕ_0 the initial condition $\overline{\boldsymbol{\Phi}}^0$ in a given cell is given by

$$\overline{\boldsymbol{\Phi}}^0(i_1, \ldots, i_D) = \Delta\mathcal{D}^{-1} \int_{\Omega(i_1,\ldots,i_D)} \phi_0(\boldsymbol{x}) \, \mathrm{d}V(\boldsymbol{x}).$$

A midpoint approximation of this integral is simply the value of ϕ_0 at the centre of $\Omega(i_1, \ldots, i_D)$, that is

$$\overline{\boldsymbol{\Phi}}^0(i_1, \ldots, i_D) \approx \phi_0 \left(x_1^{min} + \left(i_1 - \frac{1}{2} \right) \Delta x_1, \ldots, x_D^{min} + \left(i_D - \frac{1}{2} \right) \Delta x_D \right). \tag{5.7}$$

Recall that the coefficient tensors are defined in terms of the boundary integral functions W_d and V_d where for each d;

$$W_d(i_1, \ldots, i_D) = \int_{\partial_d\Omega(i_1,\ldots,i_D)} w_d(\boldsymbol{x}) \, \mathrm{d}S(\boldsymbol{x}) \quad \text{and} \quad V_d(i_1, \ldots, i_D) = \int_{\partial_d\Omega(i_1,\ldots,i_D)} v_d(\boldsymbol{x}) \, \mathrm{d}S(\boldsymbol{x})$$

for all $(i_1, \ldots, i_D)$. For each $d \in \{1, \ldots, D\}$ and $(i_1, \ldots, i_D)$ define the index functions

$$\omega_d(i_1, \ldots, i_D) := w_d \left(x_1^{min} + \left(i_1 - \frac{1}{2} \right) \Delta x_1, \ldots, x_d^{min} + i_d \Delta x_d, \ldots, x_D^{min} + \left(i_D - \frac{1}{2} \right) \Delta x_D \right), \tag{5.8}$$

$$\nu_d(i_1,\ldots,i_D) := v_d\left(x_1^{min}+\left(i_1-\frac{1}{2}\right)\Delta x_1,\ldots,x_d^{min}+i_d\Delta x_d,\ldots,x_D^{min}+\left(i_D-\frac{1}{2}\right)\Delta x_D\right), \tag{5.9}$$

so that the integral functions W_d and V_d can be approximated by

$$W_d(i_1,\ldots,i_D) = \omega_d(i_1,\ldots,i_D)\prod_{\substack{k=1\\k\neq d}}^{D}\Delta x_k, \tag{5.10}$$

and

$$V_d(i_1,\ldots,i_D) = \nu_d(i_1,\ldots,i_D)\prod_{\substack{k=1\\k\neq d}}^{D}\Delta x_k. \tag{5.11}$$

The functions W_d, V_d, ω_d and ν_d are Dth-order tensors as well. However, using bold Euler scripts to denote them only makes the equations harder to read. We continue to denote them as we have been, but keep in mind that they are also tensors.

Now for each d define three new tensors

$$\widehat{\boldsymbol{\mathcal{U}}}_d := \Delta\mathcal{D}^{-1}\boldsymbol{\mathcal{U}}_d,\ \ \widehat{\boldsymbol{\mathcal{C}}}_d := \Delta\mathcal{D}^{-1}\boldsymbol{\mathcal{C}}_d \ \text{ and } \ \widehat{\boldsymbol{\mathcal{L}}}_d := \Delta\mathcal{D}^{-1}\boldsymbol{\mathcal{L}}_d.$$

By using the midpoint approximations from (5.10) and (5.11) the tensors $\widehat{\boldsymbol{\mathcal{U}}}_d, \widehat{\boldsymbol{\mathcal{C}}}_d$ and $\widehat{\boldsymbol{\mathcal{L}}}_d$ may be written as

$$\widehat{\boldsymbol{\mathcal{U}}}_d(i_1,\ldots,i_D) = \frac{\omega_d(i_d-1)}{\Delta x_d^2} - \frac{\min\{0,\nu_d(i_d-1)\}}{\Delta x_d}, \tag{5.12}$$

$$\widehat{\boldsymbol{\mathcal{C}}}_d(i_1,\ldots,i_D) = -\frac{\omega_d(i_d)+\omega_d(i_d-1)}{\Delta x_d^2} - \frac{\max\{0,\nu_d(i_d)\}}{\Delta x_d} + \frac{\min\{0,\nu_d(i_d-1)\}}{\Delta x_d}, \tag{5.13}$$

$$\widehat{\boldsymbol{\mathcal{L}}}_d(i_1,\ldots,i_D) = \frac{\omega_d(i_d)}{\Delta x_d^2} + \frac{\max\{0,\nu_d(i_d)\}}{\Delta x_d}. \tag{5.14}$$

These hold for all $(i_1,...,i_D)$ that correspond to control volumes that are not affected by the boundary conditions. For $(i_1,...,i_D)$ corresponding to control volumes involved in the application of boundary conditions these coefficients might be different. Finally, define $\widehat{\boldsymbol{\mathcal{C}}}$ to be the sum of all $\widehat{\boldsymbol{\mathcal{C}}}_d$, and

$$\widehat{\boldsymbol{\mathcal{M}}} := \mathbf{1} + \Delta t\widehat{\boldsymbol{\mathcal{C}}}.$$

Then the finite volume method is implemented by the recurrence

$$\overline{\boldsymbol{\Phi}}^{n+1} = \widehat{\boldsymbol{\mathcal{M}}} * \overline{\boldsymbol{\Phi}}^{n} + \Delta t\sum_{d=1}^{D}\left(\tau_{d,-1}^{circ}[\widehat{\boldsymbol{\mathcal{U}}}_d * \overline{\boldsymbol{\Phi}}^{n}] + \tau_{d,1}^{circ}[\widehat{\boldsymbol{\mathcal{L}}}_d * \overline{\boldsymbol{\Phi}}^{n}]\right), \tag{5.15}$$

with initial condition $\overline{\boldsymbol{\Phi}}^{0}$ given by (5.7).

A benefit of writing this finite volume method in tensor form is that zero flux boundary conditions and Periodic boundary conditions are easy to implement. Zero flux boundary conditions are enforced by using the following coefficient tensors

$$\widehat{\boldsymbol{\mathcal{U}}}_d(i_1,i_2) := \begin{cases} 0 & \text{if } i_d = 1, \\ \frac{W_d(i_d-1)}{\Delta x_d} - \min\{0, V_d(i_d-1)\} & \text{otherwise;} \end{cases}$$
$$\widehat{\boldsymbol{\mathcal{L}}}_d(i_1,i_2) := \begin{cases} 0 & \text{if } i_d = I_d, \\ \frac{W_d(i_d)}{\Delta x_d} + \max\{0, V_d(i_d)\} & \text{otherwise;} \end{cases}$$

$$\widehat{\boldsymbol{\mathcal{C}}}_d(i_1,I_2) := \begin{cases} -\frac{W_d(i_d)}{\Delta x_d} - \max\{0, V_d(i_d)\} & \text{if } i_d = 1, \\ -\frac{W_d(i_d)+W_d(i_d-1)}{\Delta x_d} - \max\{0, V_d(i_d)\} + \min\{0, V_d(i_d-1)\} & \text{if } 1 < i_d < I_d, \\ -\frac{W_d(i_d-1)}{\Delta x_d} + \min\{0, V_d(i_d-1)\} & \text{if } i_d = I_d. \end{cases}$$

Periodic boundary conditions require that $\phi(t, x_d = x_d^{min}) = \phi(t, x_d = x_d^{max})$. Intuitively this means that the domain is viewed as a higher dimensional torus. This boundary condition can be enforced by viewing $\Omega(I_d)$ and $\Omega(1)$ as neighbouring control volumes for each $d \in \{1 \dots D\}$. In this case we would take the coefficient tensors

$$\widehat{\boldsymbol{\mathcal{U}}}_d(i_1,i_2) := \begin{cases} \frac{W_d(I_d)}{\Delta x_d} - \min\{0, V_d(I_d)\} & \text{if } i_d = 1, \\ \frac{W_d(i_d-1)}{\Delta x_d} - \min\{0, V_d(i_d-1)\} & \text{otherwise;} \end{cases}$$

$$\widehat{\boldsymbol{\mathcal{L}}}_d(i_1,i_2) := \begin{cases} \frac{W_d(1)}{\Delta x_d} + \max\{0, V_d(1)\} & \text{if } i_d = I_d, \\ \frac{W_d(i_d)}{\Delta x_d} + \max\{0, V_d(i_d)\} & \text{otherwise;} \end{cases}$$

$$\widehat{\boldsymbol{\mathcal{C}}}_d(i_1,I_2) := \begin{cases} -\frac{W_d(1)+W_d(I_d)}{\Delta x_d} - \max\{0, V_d(1)\} + \min\{0, V_d(I_D)\} & \text{if } i_d = 1, \\ -\frac{W_d(i_d)+W_d(i_d-1)}{\Delta x_d} - \max\{0, V_d(i_d)\} + \min\{0, V_d(i_d-1)\} & \text{if } 1 < i_d < I_d, \\ -\frac{W_d(1)+W_d(I_d-1)}{\Delta x_d} - \max\{0, V_d(1)\} + \min\{0, V_d(I_d-1)\} & \text{if } i_d = I_d. \end{cases}$$

Once the boundary conditions have been accounted for entries of the initial condition $\overline{\boldsymbol{\Phi}}^0$, and the coefficient tensors $\boldsymbol{\mathcal{U}}_d, \boldsymbol{\mathcal{C}}_d$ and $\boldsymbol{\mathcal{L}}_d$ are all given by functions that can be evaluated easily. So, these tensors can be formed quickly and efficiently using TT cross approximations. There is, however, no guarantee that these tensors will have small TT ranks and as a result using them in tensor train computations might be slow. To combat this, we could round the coefficient tensors to have all TT ranks equal to 1. Tensors with TT ranks equal to 1 correspond to tensors with canonical rank 1, and such tensors are rare. For this reason, rounding the coefficient tensors to have all TT ranks equal to 1 is likely to be inaccurate. It is possible to make better approximations, but these

approximations depend on the structure of the convection velocities and the diffusion coefficients. For now, the coefficient tensors are not rounded. In Chapter 6 the effect of rounding the coefficient tensors for a specific example will be considered.

For all $d \in \{1, \dots, D\}$ let $W_{d,c}$ and $V_{d,c}$ be the largest number of operations needed to evaluate a w_d and v_d, respectively . From the discussion of the cross approximation in Section 3.2.2.3 it follows that the TT-cross approximation of $\widehat{\boldsymbol{\mathcal{U}}}_d$ requires at most

$$\mathcal{O}(W_{d,c}DIR^2 + DIR^3)$$

operations. Similarly, the cross approximation of $\widehat{\boldsymbol{\mathcal{L}}}_d$ requires at most

$$\mathcal{O}(V_{d,c}DIR^2 + DIR^3)$$

operations. Therefore, the total cost of computing and rounding the tensors $\widehat{\boldsymbol{\mathcal{U}}}_d$ and $\widehat{\boldsymbol{\mathcal{L}}}_d$ for all d is

$$\mathcal{O}\big((W_{d,c} + V_{d,c})D^2IR^2 + D^2IR^3\big). \tag{5.16}$$

After adding all of the $\widehat{\boldsymbol{\mathcal{C}}}_d$s the TT ranks of $\widehat{\boldsymbol{\mathcal{C}}}$ could be suboptimal, but obtaining an estimate for the exact rank is not possible without making more assumptions about the diffusion coefficients and the convection velocities. Rounding of the tensor $\widehat{\boldsymbol{\mathcal{C}}}$ is also considered in Chapter 6. The total number of operations required for computing the $\widehat{\boldsymbol{\mathcal{C}}}_d$s by a TT-cross approximation and adding them is at most

$$\mathcal{O}((W_c + V_c)D^2IR^2 + D^2IR^3\big),$$

where

$$W_c := \max d \in \{1, \dots, D\}\{W_{d,c}\} \quad \text{and} \quad V_c := \max d \in \{1, \dots, D\}\{V_{d,c}\}.$$

Forming $\widehat{\boldsymbol{\mathcal{M}}}$ by adding all $\widehat{\boldsymbol{\mathcal{C}}}_d$s to a tensor of all ones has negligible cost. This is because addition without rounding in the TT format is a simple concatenation of cores.

Recall from (4.36) that

$$\Delta t = T\Big\lceil T \max_{i_1,\dots,i_D} \{-\widehat{\boldsymbol{\mathcal{C}}}(i_1, \dots, i_D)\}\Big\rceil^{-1},$$

gives the largest time step-size that preserves positivity as well as dividing $[0, T]$ into an integer number of equally sized intervals. The number of such intervals is N_t where

$$N_t = \frac{T}{\Delta t} = \lceil T \max_{i_1,\dots,i_D} \{-\widehat{\boldsymbol{\mathcal{C}}}(i_1, \dots, i_D)\}\Big\rceil.$$

Computing the largest entry in $-\widehat{\boldsymbol{\mathcal{C}}}$ is done using an algorithm from the TT-toolbox, [45], called `tt_max_abs`. This algorithm is an extension of the a matrix method from [14, Sec. 3] to higher order tensors and it requires at most $\mathcal{O}(DIR^2)$ operations [42].

After computing Δt all of the tensors $\widehat{\boldsymbol{\mathcal{U}}}_d$ and $\widehat{\boldsymbol{\mathcal{L}}}_d$ are multiplied by Δt. Doing so removes the need to perform a scalar multiplication at the end of each time step. By Proposition 3.2.11.1 each of these scalar multiplications require at most $\mathcal{O}(IR^2)$ operations. There are $2D$ of these multiplications that need to be computed, so in total

these multiplications require $\mathcal{O}(DIR^2)$ operations.

To finish off the initial setup, an approximate initial condition $\overline{\phi}^0$ needs to be computed. If φ_c is the number of operations needed for one evaluation of ϕ_0 then creating a cross approximation of $\overline{\boldsymbol{\Phi}}^0$ requires at most $\mathcal{O}(\varphi_c DIR^2) + \mathcal{O}(DIR^3)$ operations.

Algorithm 3 gives formal pseudo code for the initial setup procedure. In total, the number of operations required for setting up the coefficient tensors, computing Δt and computing the initial condition is at most

$$\mathcal{O}\big((W_c + V_c)D^2IR^2 + \varphi_c DIR^2 + D^2IR^3\big). \tag{5.17}$$

5.1.2 Integration Step

After setting up the coefficient tensors and the initial value tensor all that is left to do is compute

$$\overline{\boldsymbol{\Phi}}^{n+1} = \widehat{\boldsymbol{\mathcal{M}}} * \overline{\boldsymbol{\Phi}}^n + \sum_{d=1}^{D} \left(\tau_{d,-1}^{circ}[\widehat{\boldsymbol{\mathcal{U}}}_d * \overline{\boldsymbol{\Phi}}^n] + \tau_{d,1}^{circ}[\widehat{\boldsymbol{\mathcal{L}}}_d * \overline{\boldsymbol{\Phi}}^n] \right), \tag{5.18}$$

for every $n \in \{0, \dots, N_t - 1\}$. Note the Δt factor in front of the sum is dropped, because it has been absorbed into the coefficient tensors.

To generate $\overline{\boldsymbol{\Phi}}^{n+1}$ from $\overline{\boldsymbol{\Phi}}^n$ requires $2D + 1$ Hadamard products and $2D$ circular shifts to be performed. There is currently no circular shifting algorithm in the TT toolbox, however it is not hard to create one. Circular shifting an index of a tensor in TT format comes down to a circular shifting of the corresponding core.

Proposition 5.1.2
Suppose $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \dots \times I_N}$ is given in TT format with cores $\boldsymbol{\mathcal{X}}_1, \dots, \boldsymbol{\mathcal{X}}_N$. For any integers $m \in \{1, \dots, N\}$ and $k \in \{-I_n + 1, \dots, -1, 1, \dots, I_n - 1\}$ the tensor $\widehat{\boldsymbol{\mathcal{X}}} := \tau_{m,k}^{circ}[\boldsymbol{\mathcal{X}}]$ has a TT decomposition with cores

$$\widehat{\boldsymbol{\mathcal{X}}}_n := \begin{cases} \boldsymbol{\mathcal{X}}_n & \text{if } n \neq m, \\ \tau_{2,k}[\boldsymbol{\mathcal{X}}_n] & \text{if } n = m. \end{cases} \tag{5.19}$$

Proof:
Fix an arbitrary tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \dots \times I_N}$ and suppose it is given in TT format with cores $\boldsymbol{\mathcal{X}}_1, \dots, \boldsymbol{\mathcal{X}}_N$. Pick any $m = 1, \dots, N$ and $k \in \{-I_m + 1, \dots, I_m - 1\}$ and define $\widehat{\boldsymbol{\mathcal{X}}} := \tau_{m,k}^{circ}[\boldsymbol{\mathcal{X}}]$. Let the index $\tau(i_m)$ be given by

$$\tau(i_m) := (i_m - k) \bmod I_m.$$

From the definition of the circular shifting operator we find that for all $i_1, \dots, i_N$;

$$\begin{aligned} \widehat{\boldsymbol{\mathcal{X}}} &= \boldsymbol{\mathcal{X}}(i_1, \dots, i_{m-1}, \tau(i_m), i_{m+1}, \dots, i_N) \\ &= \boldsymbol{\mathcal{X}}_1(:, i_1, :) \dots \boldsymbol{\mathcal{X}}_{m-1}(:, i_{m-1}, :) \boldsymbol{\mathcal{X}}_m(:, \tau(i_m), :) \boldsymbol{\mathcal{X}}_{m+1}(:, i_{m+1}, :) \dots \boldsymbol{\mathcal{X}}_N(:, i_N, :) \\ &= \boldsymbol{\mathcal{X}}_1(:, i_1, :) \dots \boldsymbol{\mathcal{X}}_{m-1}(:, i_{m-1}, :) \tau_{2,k}^{circ}[\boldsymbol{\mathcal{X}}_m](:, i_m, :) \boldsymbol{\mathcal{X}}_{m+1}(:, i_{m+1}, :) \dots \boldsymbol{\mathcal{X}}_N(:, i_N, :), \end{aligned}$$

which gives the result. □

Algorithm 3: Tensor Train FVM set up

Input: Dimension of the problem D. Initial condition function ϕ_0. Diffusion coefficient $\boldsymbol{w}$. Convection velocity $\boldsymbol{v}$. Domain bounds x_d^{min} and x_d^{max}, and number of control volumes I_d for each dimension $d = 1, \dots, D$. Final time T.

Result: The intital condition $\overline{\boldsymbol{\Phi}}^0$ and the coefficient tensors $\widehat{\boldsymbol{\mathcal{M}}}$ in TT format, $\widehat{\boldsymbol{\mathcal{U}}}_d$, $\widehat{\boldsymbol{\mathcal{L}}}_d$ for each d, the time step-size Δt and the number of time steps N_t.

Initialisation: .

For each $d \in \{1, \dots, D\}$ set

$$\Delta x_d = \frac{x_d^{max} - x_d^{min}}{I_d}.$$

For each $d \in \{1, \dots, D\}$ define the discrete diffusion coefficient ω_d and convection velocity ν_d according to (5.8) and (5.9), respectively.

Initialise $\widehat{\boldsymbol{\mathcal{C}}} \in \mathbb{R}^{I_1 \times \dots \times I_D}$ with all entries equal to zero.

Main Algorithm:

{Approximate initial condition}

Compute TT-cross approximation of $\overline{\boldsymbol{\Phi}}^0$ with entries given by (5.7).

{Approximate central coefficient tensor}

for $d = 1, \dots, D$ **do**

  Compute a TT-cross approximation of $\widehat{\boldsymbol{\mathcal{C}}}_d$ according to (5.13).

  $\widehat{\boldsymbol{\mathcal{C}}} = \widehat{\boldsymbol{\mathcal{C}}} + \widehat{\boldsymbol{\mathcal{C}}}_d$.

end

$N_t = \left\lceil T \max_{i_1,\dots,i_D} \{-\widehat{\boldsymbol{\mathcal{C}}}(i_1, \dots, i_D)\} \right\rceil$.

$\Delta t = T/Nt$.

$\widehat{\boldsymbol{\mathcal{M}}} = \mathbf{1} + \Delta t \widehat{\boldsymbol{\mathcal{C}}}$.

{Approximate off-centre coefficient tensors}

for $d = 1, \dots, \mathrm{d}$ **do**

  Compute a TT-cross approximation of $\widehat{\boldsymbol{\mathcal{U}}}_d$ according to (5.12).

  Redefine $\widehat{\boldsymbol{\mathcal{U}}}_d = \Delta t \widehat{\boldsymbol{\mathcal{U}}}_d$.

  Compute a TT-cross approximation of $\widehat{\boldsymbol{\mathcal{L}}}_d$ according to (5.14).

  Redefine $\widehat{\boldsymbol{\mathcal{L}}}_d = \Delta t \widehat{\boldsymbol{\mathcal{L}}}_d$.

end

Return: $\overline{\boldsymbol{\Phi}}^0$, $\widehat{\boldsymbol{\mathcal{M}}}$, all $\widehat{\boldsymbol{\mathcal{U}}}_d$'s, all $\widehat{\boldsymbol{\mathcal{L}}}_d$'s, Δt and N_t.

Proposition 5.1.2 gives a simple algorithm for circular shifting of indices in the TT format that will have negligible computational complexity. And more importantly, circular shifting doesn't change the TT ranks. Algorithm 4 gives formal pseudo code for the circular shifting in the TT format. We call this algorithm the TT-circshift algorithm. From now on when we refer to circular shifting of tensors in the TT format it is assumed that this algorithm is used.

Algorithm 4: TT-circshift

Input: An Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ in TT format, a positive integer $m \in \{1, \ldots, N\}$ and another integer $k \in \{-I_n + 1, \ldots, -1, 1, \ldots, I_n - 1\}$.
Result: The tensor $\tau_{m,k}[\boldsymbol{\mathcal{X}}]$ in TT format.
Initialisation: Let $\boldsymbol{\mathcal{X}}_1, \ldots, \boldsymbol{\mathcal{X}}_N$ be the TT cores of $\boldsymbol{\mathcal{X}}$.
Main Algorithm:
1 Define $\widehat{\boldsymbol{\mathcal{X}}}_m$ by $\widehat{\boldsymbol{\mathcal{X}}}_m := \tau_{2,k}[\boldsymbol{X}_m]$.
2 **Return**: $\tau_{2,k}[\boldsymbol{X}]$ in TT format with cores $\boldsymbol{\mathcal{X}}_1, \ldots, \boldsymbol{\mathcal{X}}_{m-1}, \widehat{\boldsymbol{\mathcal{X}}}_m, \boldsymbol{\mathcal{X}}_{m+1}, \ldots, \boldsymbol{\mathcal{X}}_N$.

The Hadamard products in (5.18) each require $\mathcal{O}(DIR^2\mathcal{R})$ where $\mathcal{R}$ is the maximum TT rank of $\overline{\boldsymbol{\Phi}}^n$. It is clear that the TT ranks of the tensors involved in the computations will be one of the biggest factors in the computational complexity of the overall method. This will be especially true if the exact analytic solution to the convection-diffusion equation is a function with TT ranks that increase over time. Such solutions occur if discontinuities develop over time. However, if the solution remains smooth over time one may assume that the TT ranks remain constant, or at least bounded.

Assume that the TT ranks of $\overline{\boldsymbol{\Phi}}^0$ are bounded by $\mathcal{R}_0$, and for some $n \in \{0, \ldots, N_t - 1\}$ the approximation $\overline{\boldsymbol{\Phi}}^n$ has TT ranks that are also bounded by $\mathcal{R}_0$. Then computing $\overline{\boldsymbol{\Phi}}^{n+1}$ by (5.18) will require at most $\mathcal{O}(D^2 I \mathcal{R}_0^2 R^2)$ operations. If $\overline{\boldsymbol{\Phi}}^{n+1}$ is not rounded its TT ranks will be of the order $\mathcal{O}(D\mathcal{R}_0)$ and the approximation at the next time step will have TT ranks of the order $\mathcal{O}(D^2\mathcal{R}_0)$.To prevent exponential rank growth round $\overline{\boldsymbol{\Phi}}^{n+1}$ to have TT ranks no higher than $\mathcal{R}_0$, unless $\mathcal{R}_0 = 1$. If $\mathcal{R}_0 = 1$ then we round $\overline{\boldsymbol{\Phi}}^{n+1}$ to have TT ranks no higher than D.

Recall that rounding a tensor in TT format requires $\mathcal{O}(DIR^3)$ operations, so since $\overline{\boldsymbol{\Phi}}^{n+1}$ has TT ranks of the order $\mathcal{O}(D\mathcal{R}_0)$, rounding it wil require at most $\mathcal{O}(D^4 I \mathcal{R}_0^3)$ operations. Therefore, the total cost of computing $\overline{\boldsymbol{\Phi}}^{n+1}$ from $\overline{\boldsymbol{\Phi}}^n$ is

$$\mathcal{O}\big(D^4 I \mathcal{R}_0^3 + D^2 I \mathcal{R}_0^2 R^2\big).$$

There are N_t of these $\overline{\boldsymbol{\Phi}}^n$ that need to be computed, which means that the total cost of computing $\overline{\boldsymbol{\Phi}}^{N_t}$ from $\overline{\boldsymbol{\Phi}}^0$ is

$$\mathcal{O}\big(N_t D^4 I \mathcal{R}_0^3 + N_t D^2 I \mathcal{R}_0^2 R^2\big). \tag{5.20}$$

Algorithm 5 gives the formal pseudo code for the time stepping procedure. By combining (5.20) and (5.17) an estimate of the computational cost of entire TT format finite volume method can be found.

Theorem 5.1.3
Suppose that a function ϕ satisifies

$$\begin{cases} \frac{\partial \phi}{\partial t}(t, \boldsymbol{x}) - \nabla \cdot [\boldsymbol{w}(\boldsymbol{x}) * \nabla \phi(t, \boldsymbol{x})] + \nabla \cdot [\phi(t, \boldsymbol{x}) \boldsymbol{v}(\boldsymbol{x})] = 0 & \forall \boldsymbol{x} \in \mathbb{R}^D,\ t > 0 \\ \phi(0, \boldsymbol{x}) = \phi_0(\boldsymbol{x}), & \forall \boldsymbol{x} \in \mathbb{R}^D. \end{cases}$$

Fix any $T > 0$ and a bounded rectangular domain

$$\Omega := \bigtimes_{d=1}^{D} (x_d^{min}, x_d^{max}).$$

Then given any positive integers $I_1, \ldots, I_D$ denoting the number of control volumes along each dimension Algorithm 5 generates a finite volume approximation of $\phi(T, \boldsymbol{x})$ in Ω, with the approximation given in the TT format. The total computational complexity is

$$\mathcal{O}\big(N_t D^4 I \mathcal{R}_0^3 + N_t D^2 I \mathcal{R}_0^2 R^2 + \varphi_c D I \mathcal{R}_0^2 + (W_c + V_c) D^2 I R^2\big) \tag{5.21}$$

Algorithm 5: TT-FVM

Input: Dimension of the problem D. Initial condition function ϕ_0. Diffusion coefficient $\boldsymbol{w}$. Convection velocity $\boldsymbol{v}$. Domain bounds x_d^{min} and x_d^{max}, and number of control volumes I_d for each dimension $d = 1, \ldots, D$. Final time T.

Result: A finite volume approximation of of $\phi(T, 0)$ in TT format.

Initialisation: .

Use Algorithm 3 to compute the tensors $\overline{\boldsymbol{\Phi}}^0$, $\widehat{\boldsymbol{\mathcal{M}}}$, $\{\widehat{\boldsymbol{\mathcal{U}}}_d\}_{d=1}^D$, $\{\widehat{\boldsymbol{\mathcal{L}}}_d\}_{d=1}^D$ in TT format, and the scalars Δt, and N_t .

Let $\mathcal{R}_0$ be the largest TT rank of $\overline{\boldsymbol{\Phi}}^0$.

Main Algorithm:

for $n = 0, \ldots, N_t - 1$ **do**
  Set $\overline{\boldsymbol{\Phi}}^{n+1} = \overline{\boldsymbol{\Phi}}^{n+1} * \widehat{\boldsymbol{\mathcal{M}}}$.
  for $d = 1, \ldots, D$ **do**
    Compute $\boldsymbol{\mathcal{U}}' = \widehat{\boldsymbol{\mathcal{U}}}_d * \overline{\boldsymbol{\Phi}}^n$.
    Compute $\boldsymbol{\mathcal{L}}' = \widehat{\boldsymbol{\mathcal{L}}}_d * \overline{\boldsymbol{\Phi}}^n$.
    Set $\overline{\boldsymbol{\Phi}}^{n+1} = \overline{\boldsymbol{\Phi}}^{n+1} + \tau_{d,-1}^{circ}[\boldsymbol{\mathcal{U}}'] + \tau_{d,1}^{circ}[\boldsymbol{\mathcal{L}}']$.
  end
  Round $\overline{\boldsymbol{\Phi}}^{n+1}$ to have TT ranks no higher than $\mathcal{R}_0$.
end

Return: $\overline{\boldsymbol{\Phi}}^{N_t}$ in TT format.

5.2 Numerical tests

5.2.1 3D example

The TT-FVM algorithm was tested on a 3-dimensional example similar to the 2-dimensional example in Chapter 4. Suppose ϕ is a function on $\mathbb{R}^3$ and for some $k > 0$ satisfies

$$\begin{cases} \frac{\partial \phi}{\partial t} - k\frac{\partial^2 \phi}{\partial x_1^2} - k\frac{\partial^2 \phi}{\partial x_2^2} - k\frac{\partial^2 \phi}{\partial x_3^2} + x_2\frac{\partial \phi}{\partial x_1} - x_1\frac{\partial \phi}{\partial x_2} + k\frac{\partial \phi}{\partial x_2} = 0, & \forall t > 0,\ \boldsymbol{x} \in \mathbb{R}^3, \\ \phi(0, \boldsymbol{x}) = \phi_0(\boldsymbol{x}), & \forall \boldsymbol{x} \in \mathbb{R}^3. \end{cases}$$

As in the 2-dimensional case it can be shown by the method of characteristics that ϕ is a rotation and translation of a solution for the heat equation. In particular

$$\phi(t, \boldsymbol{x}) = \widehat{\phi}(t, x_1 \cos t - x_2 \sin t, x_1 \sin t + x_2 \cos t, x_3 - kt),$$

where $\widehat{\phi}$ solves

$$\begin{cases} \frac{\partial \widehat{\phi}}{\partial t} - k\frac{\partial^2 \widehat{\phi}}{\partial x_1^2} - k\frac{\partial^2 \widehat{\phi}}{\partial x_2^2} - k\frac{\partial^2 \widehat{\phi}}{\partial x_3^2} = 0, & \forall t > 0,\ \boldsymbol{x} \in \mathbb{R}^3, \\ \widehat{\phi}(0, \boldsymbol{x}) = \phi_0(\boldsymbol{x}), & \forall \boldsymbol{x} \in \mathbb{R}^3. \end{cases}$$

We took $k = 0.01$ and the initial condition

$$\phi_0(\boldsymbol{x}) = \psi(1.5, x_1 - 0.2)\psi(1.5, x_2)\psi(1.5, x_3),$$

where $\psi(t, x)$ is the fundamental solution of the 1-dimensional heat equation. The solution ϕ was approximated on $(-1, 1)^3$ at $t = \pi/4$ with a uniform $N \times N \times N$ grid of control volumes inside the domain. For boundary conditions zero-flux boundary conditions were chosen. The results from the TT-FVM algorithm were also compared with a standard matrix method for N ranging from 30 to 250. This range is limited because the matrix method started running out of memory on the machine used. Figure 5.2 summarises the results of this comparison. It should be noted, that for this case the L^2 error instead of the L^1 error was used as a measure of convergence. This was done because computing L^1 errors in TT format is difficult and computationally expensive, while the L^2 error can be found by the Frobenius norm.

Figure 5.2a and Figure 5.2b give evidence that both methods converge to the exact solution as the grid is refined. Moreover, the TT-FVM converges slightly slower than the usual matrix method. This is probably due to the fact that the TT format is not an exact representation of the tensors involved in the finite volume method. Rounding the TT approximation after each time step will also be slowing down convergence. However, the convergence is not so much slower that it will have any practical implication, especially when considering the difference in computation times. At $N = 250$ the standard matrix method took over 10 minutes to run while the TT-FVM took less than a minute. The improvement in computation time is seen even better when considering the asymptotic orders of the computation times against N. For this example, the computation time for the standard matrix method scaled as $\mathcal{O}(N^{3.334})$, while the computation time for the TT-FVM algorithm scaled as $\mathcal{O}(N^{1.8285})$. To find better approximations of the orders of the computation times this comparison would have to

be run for a much larger range of N. However, the purpose of this example is to illustrate the improvement made in computation times by implementing the finite volume method in the TT-format. Finally, in Figure 5.2e and Figure 5.2f it is seen that the TT format speeds up both the initial setup and the integration step. The setup time was nearly constant in the TT-format for this example, which greatly improves computation time. The time spent on the integration step increases mainly because the number of time steps, N_t, increases as the number of control volumes increases. In fact, in this example it was measured that N_t grew as $\mathcal{O}(N^{1.6012})$ (See Figure5.1). The growth in N_t is a direct result of the fact that Δt is shrinking in order to preserve positivity. As this is a crucial to property of the finite volume method that we used, all algorithms that implement it will have this growth in N_t.

5.2.2 Growth of computation time with dimension

Now that we have evidence that the TT-FVM beats the standard matrix method for implementing the finite volume method it is also good to look at how the computation time is affected by an increase in dimension. Consider the IVP

$$\begin{cases} \frac{\partial \phi}{\partial t} + k \sum_{d=1}^{D} \left[-\frac{\partial^2 \phi}{\partial x_d^2} + \frac{\partial \phi}{\partial x_d} \right] = 0, & \forall t > 0,\ \boldsymbol{x} \in \mathbb{R}^D, \\ \phi(0, \boldsymbol{x}) = \phi_0(\boldsymbol{x}), & \forall \boldsymbol{x} \in \mathbb{R}^D. \end{cases}$$

The solution of this IVP was approximated at $t = 0.5$ for $k = 0.01$ on $(-1, 1)^D$ for increasing D using a uniform grid of 100 control volumes along each dimension. The initial condition that is used is

$$\phi_0(\boldsymbol{x}) = \prod_{d=1}^{D} \psi(1.5, x_d).$$

The TT-FVM algorithm with zero flux boundary conditions was on this problem for D ranging from 3 to 30. Figure 5.3 shows a summary of the results. In Figure 5.3a it is shown that the total computation time grew as $\mathcal{O}(D^{4.3478})$, which is slightly bigger than the order of growth predicted in Theorem 5.1.3. However, this does not mean that Theorem 5.1.3 is incorrect. The computational complexity given in (5.21) does not take into account that the other variables such as the ranks R and number of time steps N_t may depend on the dimension. In fact, Figure 5.3d and Figure 5.3e shows that the maximum TT ranks of the coefficient tensors and the number of time steps both grew approximately linearly with dimension. Taking this into account, Theorem 5.1.3 would predict that the total computation time would grow as $\mathcal{O}(D^6)$. Figure 5.3b and Figure 5.3c show that the main driver of the growth in computation time is the the process of time-stepping. For higher dimensions one can expect the setup time to become negligible when compared to the time spent on actually computing $\overline{\boldsymbol{\Phi}}^{N_t}$. So, if one would try to speed up the TT-FVM algorithm this is where the energy would need to be focussed.

Using standard methods where every entry in the discrete approximation of $\overline{\boldsymbol{\Phi}}^n$ is computed explicitly the computation times would scale exponentially in dimension. So, while our method does not scale linearly with dimension it does beat the exponential growth of current methods. This means that our TT-FVM algorithm could be used to take on higher dimensional problems, one would only need to be careful to ensure that the coefficient functions will not induce coefficient tensors of high rank.

Figure 5.1: Number of time steps increases with number of control volumes

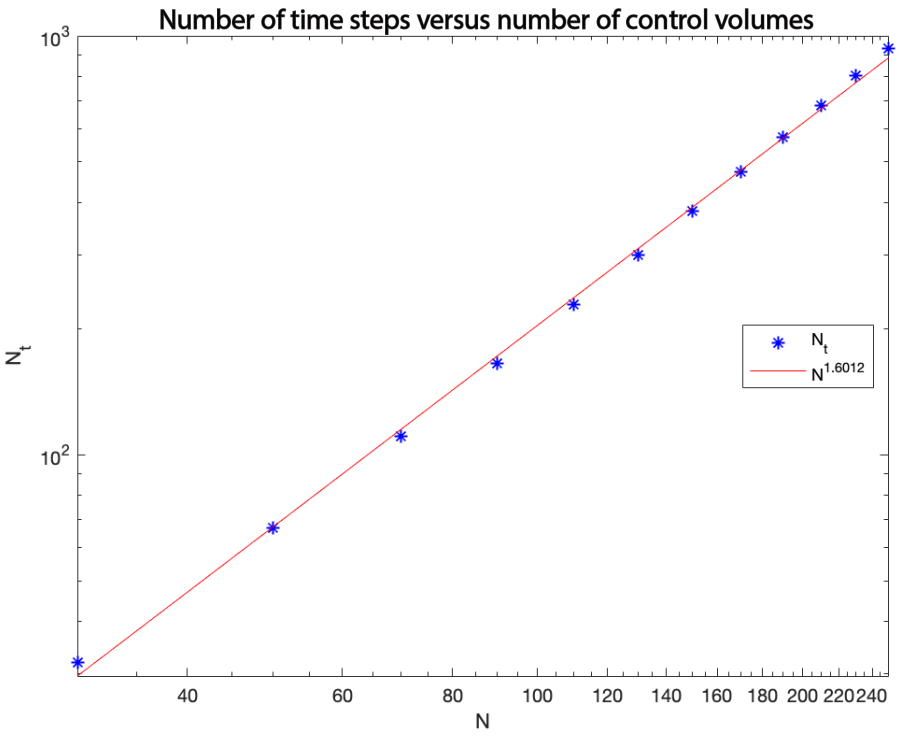
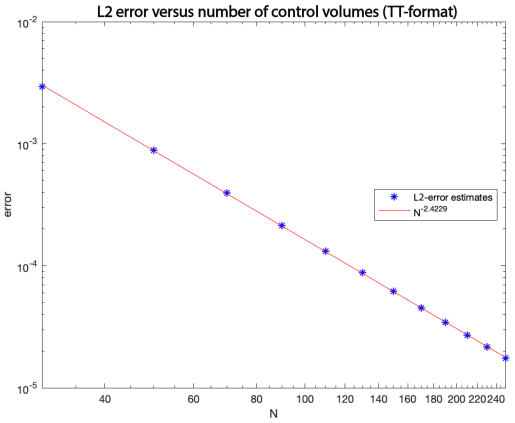
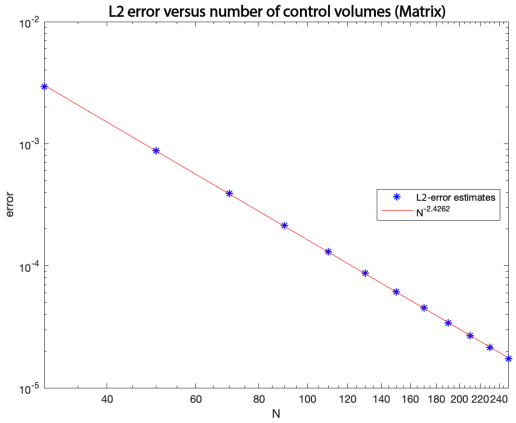


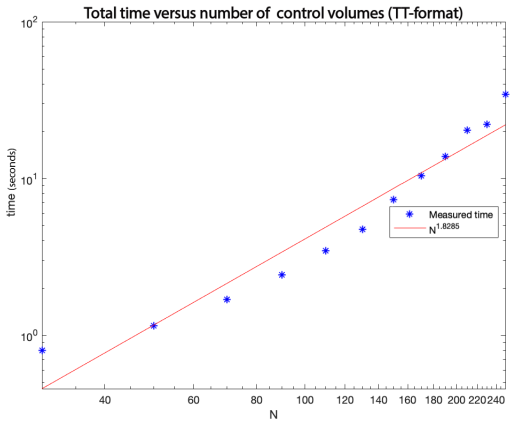
Figure 5.2: Summary Plots of 3D example



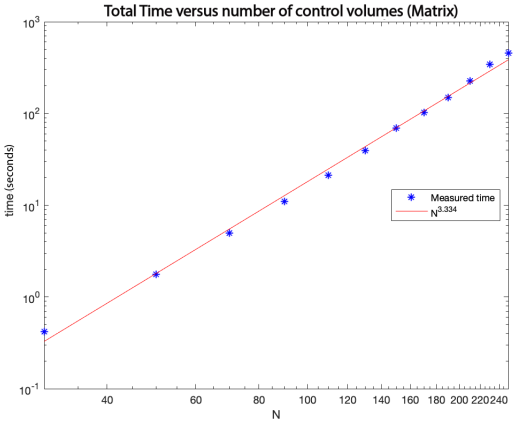
(a) Global L^2 error of the TT-FVM algorithm.



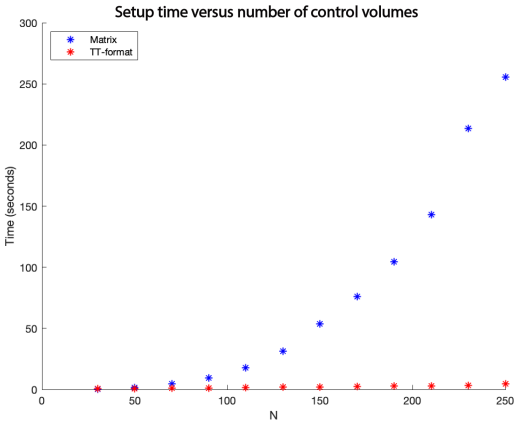
(b) Global L^2 error of the standard matrix method.



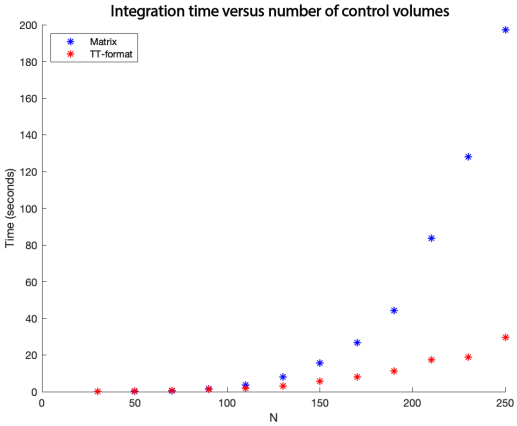
(c) Total computation time of the TT-FVM.



(d) Total computation time of the standard matrix method.

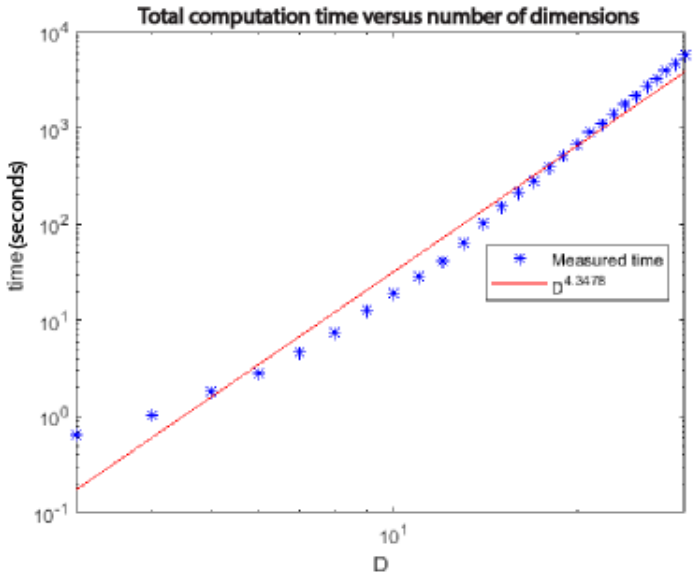


(e) Setup computation times

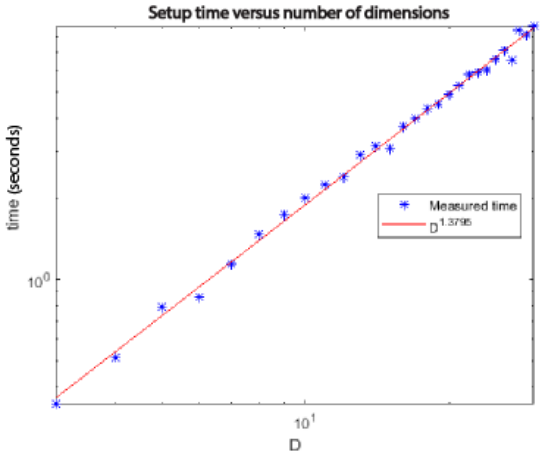


(f) Time stepping computation times.

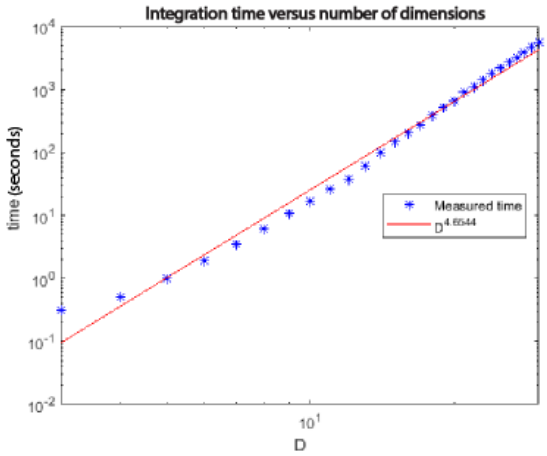
Figure 5.3: Summary plots for TT-FVM with increasing dimensions



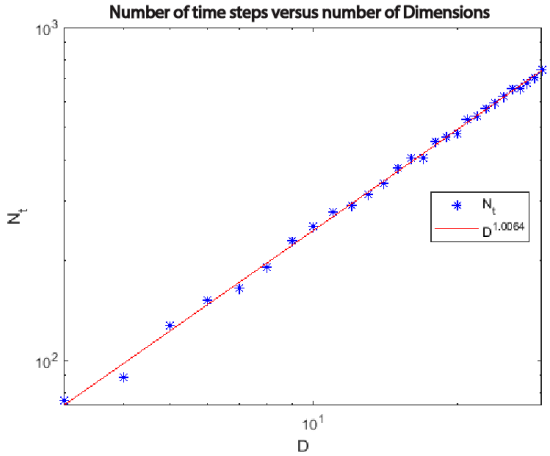
(a) Total Computation time.



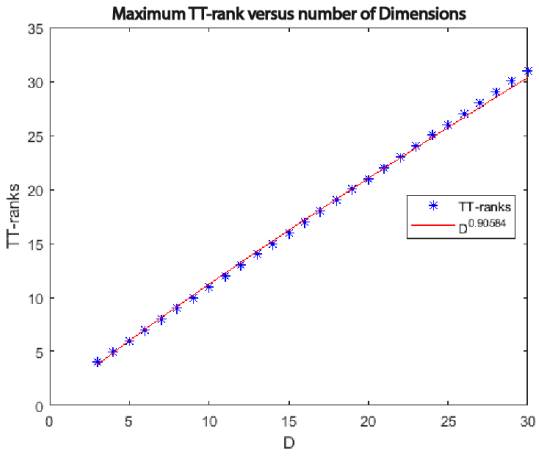
(b) Setup time.



(c) Computation time for the integration step.



(d) Number of time steps



(e) Maximum TT-ranks of Coefficient Tensors.

Chapter 6

Case Study: Approximating the Allele Frequency Spectrum

So far, all of the problems that we solved with our finite volume method and the TT-FVM algorithm had known analytic solutions. Applying our methods to these problems was important to test whether the methods are in fact working. In this chapter the TT-FVM algorithm is applied to a problem from population genetics for which no general analytic solution is known. In particular we approximate an array (or tensor) known as the *Allele Frequency Spectrum*.

6.1 Background Genetics

In this section, a brief overview of the key terms from genetics is given. The main purpose of this overview is to provide all the necessary information required for the definition of the allele frequency spectrum.

We start at the level of *DNA* (deoxyribonucleic acid). DNA holds the genetic information of an organism. A DNA molecule is made of two strands, wound around each other in a right-handed helix. Each strand consists of a sequence of four *nucleotides*, adenine (A) , thymine (T) , guanine (G) and cytosine (C). Weak chemical bonds form between A and T, and between G and C at corresponding positions in opposite strands (see [20, p. 2-3]). These weak bonds hold the two strands together. Because of these bonds, if one strand contains an A in a given position, the opposite strand contains a T in the same position. Similarly if one strand contains a G in a given position, the opposite strand contains a C. For this reason, we call A-T and G-C *base pairs*.

An organism's *genome* is the full sequence of base pairs that carries the genetic information of that organism. Usually, the genome does not consist of just one single DNA molecule. Instead, the genome is broken up into discrete units, each of which is a DNA molecule. These discrete units are called *chromosomes*. In humans, each cell has 46 individual chromosomes, but they exist in pairs. So, the 46 chromosomes are divided into two sets of 23 chromosomes, one set from each parent. Not all organisms have two sets of chromosomes in their cells, some have more. The *ploidy* of an organism is the number of complete chromosome sets in a cell. For example, humans have a ploidy of two. Organisms with a ploidy of two are called *diploid* (see Sections 6.3, 12.1 and 15.4 of [62]).

Certain parts of the genome encode *genes*, these form the genetic information passed from parents to their offspring. Physical traits such as eye colour are encoded by genes. The specific position of a gene along the genome is called its *locus*, [62, S. 13.1]. At some loci (plural of locus) different base pair sequences can appear. Such sequences encode different forms of a gene. The different forms of a gene that can occur at a locus are called the *alleles* at that locus, [20, p. 2]. If two or more alleles can appear at a locus, then that locus is called a *polymorphic site*, [22, p. 104]. A polymorphic site at which there are only two possible alleles is called *diallelic*.

Each individual in a single population or species will have their own genome. Now, consider P distinct populations of the same species, and from each population sample $n_1, \ldots, n_P$ individuals. We would have $N_p = \rho n_p$ genomic sequences from population p, where ρ is the *ploidy* of the species in question. If the ancestral state of the species' genome is known then an *allele frequency spectrum* (AFS) can be calculated, which we denote by $\boldsymbol{\mathcal{M}}$ (Not to be confused with the coefficient tensor $\boldsymbol{\mathcal{M}}$ from the finite volume method). The allele frequency spectrum is an $N_1 \times \ldots \times N_p$ time dependent tensor which records the number of *diallelic polymorphic sites* at which the *derived allele* appears,[18],[34]. At a diallelic polymorphic site the *ancestral allele* is the allele found in the ancestor's genome and the *derived allele* is the other allele that can be found at that site. So, the entry $\boldsymbol{\mathcal{M}}(t, d_1, d_2, \ldots, d_P)$ of the AFS records the number of sites at which the derived allele was found in d_1 samples in population 1 in generation t, d_2 samples in population 2 in generation t, and so on.

The model for the allele frequency spectrum that we use assumes that the species reproduces according to a *Wright-Fisher reproduction model*. This reproduction model assumes that the population has a constant size N and consists of diploid hermaphroditic individuals. This means that each individual inherited two sets of chromosomes, one from each of its parents. We say that the individuals are hermaphroditic because any two individuals can mate to produce offspring. That is, the individuals are sexless. Reproduction occurs in discrete generations, although the version of the model we use assumes that allele frequencies change continuously in time.

In the Wright-Fisher model, reproduction works as follows: At the time of reproduction, a new individual is created by randomly sampling, with replacement, two parents. At each site the new individual now inherits one allele from each of its parents, since we are dealing with a diploid population. Once N new individuals are created all of the parents from the previous generation die and only the new individuals remain. At the end of every generation, this reproduction process is repeated (see [2] and [56] for more).

6.2 Modelling the Allele Frequency Spectrum

Time evolution of the allele frequencies can be done by first modelling the time evolution a probability density $\phi(t, x_1, x_2, \ldots, x_p)$. The density ϕ is the density of derived mutations at relative frequencies $x_1, x_2, \ldots, x_p$ in populations $1, 2 \ldots, P$ at time t (all x_p's range from 0 to 1). Time t is measured in generations. From Kimura's 1964 paper [25] we know that under an infinite sites model with Wright-Fisher reproduction ϕ is

well approximated by the solutions to a second order linear partial differential equation;

$$\frac{\partial \phi}{\partial \tau}(\tau, x_1, x_2, \ldots, x_P) = \sum_{p=1}^{P} \frac{\partial^2}{\partial x_p^2}\left[\phi(\tau, x_1, x_2, \ldots, x_p)\frac{x_p(1-x_p)}{2\nu_p}\right] - \sum_{p=1}^{P} \frac{\partial}{\partial x_p}\left[\phi(\tau, x_1, x_2, \ldots, x_P) \sum_{q=1,\ q\neq p}^{P} M_{p\leftarrow q}(x_q - x_p)\right]. \tag{6.1}$$

We call this equation the *Wright-Fisher diffusion equation*. Time has been rescaled to $\tau := t/(2N_{ef})$, where N_{ef} is the effective population of the species in question. The parameters $\nu_p = N_p/N_{ef}$ are the relative effective population size of population p (see [2] for more on effective population sizes). The scaled migration rate from population q to population p is $M_{p\leftarrow q} = 2N_{ef}m_{p\leftarrow q}$, where $m_{p\leftarrow q}$ is the proportion of chromosomes per generation in population p that are new introductions from population q . If ϕ is known at any time τ, the expected value of each entry in the allele frequency spectrum at that time is given by

$$\boldsymbol{\mathcal{M}}(\tau, d_1, d_2 \ldots, d_P) = \int_{[0,1]^P} \prod_{p=1}^{P} \binom{N_p}{d_p} x_p^{d_p}(1-x_p)^{N_p - d_p}\phi(\tau, x_1, x_2 \ldots, x_P)\,\mathrm{d}V(\boldsymbol{x}), \tag{6.2}$$

where d_p is an integer that ranges from 0 to N_p for each p,[34, eq. (1.3)].

The Wright-Fisher model means that the frequency of each allele in the genomic sequence follows a certain discrete Markov chain. Under the infinite sites model this Markov chain is well approximated by a diffusion process where the state of the process is the allele frequencies in each population. The density ϕ then represents the conditional probability distribution over the state space given the initial condition and the partial differential equation (6.1) is the Kolmogorov forward equation for the diffusion process (see [7, Chap. 7] for more on the Wright-Fisher model as a diffusion process).

Approximating the AFS numerically is a two-part problem. The first and biggest part of the problem is to solve the differential equation (6.1) to find ϕ at the desired time. Second, a way to efficiently compute the entries of the AFS by evaluating the integral in (6.2) needs to be found. We solve the first part of the problem by using the TT-FVM algorithm from the previous chapter, however not without a bit of work first. The Wright-Fisher diffusion equation (6.1) is a convection diffusion equation, however it is not stated in the way that we defined such equations in Definition 4.0.1.

Proposition 6.2.1
Suppose ϕ satisfies the Wright-Fisher diffusion equation then define for all $p \in \{1, \ldots, P\}$

$$\lambda_p := \frac{1}{\nu_p} - \sum_{q=1\ q\neq p}^{P} M_{p\leftarrow q},$$

$$w_p(\boldsymbol{x}) := \frac{x_p(1-x_p)}{2\nu_p}, \quad v_p(\boldsymbol{x}) := \lambda_p x_p + \sum_{q=1,\ q\neq p} M_{p\leftarrow q}x_q - \frac{1}{2\nu_p}.$$

Then ϕ also satisfies

$$\frac{\partial \phi}{\partial \tau}(\tau, \boldsymbol{x}) - \nabla \cdot [\boldsymbol{w}(\boldsymbol{x}) * \nabla \phi(\tau, \boldsymbol{x})] + \nabla \cdot [\phi(\tau, \boldsymbol{x}) \boldsymbol{v}(\boldsymbol{x})] = 0,$$

where $\boldsymbol{w}(\boldsymbol{x}) = (w_1(\boldsymbol{x}), w_2(\boldsymbol{x}), \ldots, w_(\boldsymbol{x}))$ and $\boldsymbol{v}(\boldsymbol{x}) = (v_1(\boldsymbol{x}), v_2, \ldots, v_p(\boldsymbol{x}))$.

Proof:
To prove this proposition first define a vector field $\boldsymbol{F}$ by its components;

$$F_p(\tau, \boldsymbol{x}) := \frac{\partial}{\partial x_p} \left[\phi(\tau, \boldsymbol{x}) \frac{x_p(1 - x_p)}{2\nu_p} \right] - \phi(\tau, \boldsymbol{x}) \sum_{q=1,\ q \neq p}^{P} M_{p \leftarrow q}(x_q - x_p), \tag{6.3}$$

for all $p \in \{1, \ldots, P\}$. The Wright-Fisher Diffusion equation may be written more concisely as

$$\frac{\partial \phi}{\partial \tau}(\tau, \boldsymbol{x}) = \nabla \cdot \boldsymbol{F}(t, \boldsymbol{x}). \tag{6.4}$$

Computing the derivative in (6.3) by the product rule shows that for each $p = 1, \ldots, P$

$$\begin{aligned} F_p(\tau, \boldsymbol{x}) &= \frac{\partial \phi}{\partial x_p}(\tau, \boldsymbol{x}) \frac{x_p(1 - x_p)}{2\nu_p} + \phi(\tau, \boldsymbol{x}) \frac{1 - 2x_p}{2\nu_p} - \phi(\tau, \boldsymbol{x}) \sum_{q=1,\ q \neq p}^{P} M_{p \leftarrow q}(x_q - x_p) \\ &= \frac{\partial \phi}{\partial x_p}(\tau, \boldsymbol{x}) \frac{x_p(1 - x_p)}{2\nu_p} + \phi(\tau, \boldsymbol{x}) \left(\frac{1 - 2x_p}{2\nu_p} - \sum_{q=1,\ q \neq p}^{P} M_{p \leftarrow q}(x_q - x_p) \right). \end{aligned}$$

Define two new time-homogeneous vector fields $\boldsymbol{w}$ and $\boldsymbol{v}$ by

$$w_p(\boldsymbol{x}) := \frac{x_p(1 - x_p)}{2\nu_p} \tag{6.5}$$

$$v_p(\boldsymbol{x}) := -\frac{1 - 2x_p}{2\nu_p} + \sum_{q=1,\ q \neq p}^{P} M_{p \leftarrow q}(x_q - x_p), \tag{6.6}$$

The differential equation (6.3) can be written more verbosely as

$$\frac{\partial \phi}{\partial \tau}(\tau, \boldsymbol{x}) - \nabla \cdot [\boldsymbol{w}(\boldsymbol{x}) * \nabla \phi(\tau, \boldsymbol{x})] + \nabla \cdot [\phi(\tau, \boldsymbol{x}) \boldsymbol{v}(\boldsymbol{x})] = 0.$$

The proposition now follows by rearranging (6.6) and grouping all $x_1, \ldots, x_P$ terms according to their subscripts.

□

Now that the diffusion coefficients and the convection velocities of the Wright-Fisher diffusion equation are known the TT-FVM algorithm can be used to approximate a solution to ϕ at any time τ. All that is left to do is decide on boundary conditions. To date boundary conditions for this problem have not been completely resolved. Approximations have been suggested, but they are either not very rigorous or not very accurate (see [18] and [34]). For simplicity we will assume zero flux boundary conditions. By assuming such boundary conditions, we are allowing alleles to become fixed (present in all individuals) or lost (present in no individuals) within a single

population. Novel mutations occurring in a population would appear in ϕ as an injection of density on the boundaries at low frequencies. So, by choosing zero flux boundary conditions it is also assumed that no mutations occur.

To form a TT approximation of ϕ using the TT-FVM the control volumes and coefficient tensors need to be defined. The domain of the problem is $[0,1]^P$, which can be exactly partitioned by the rectangular control volumes used by the TT-FVM. So for each $p \in \{1\dots,P\}$ choose a positive integer I_p and define the grid spacing

$$\Delta x_p = I_p^{-1}.$$

The control volumes are

$$\Omega(i_1,\dots,i_P) = \bigtimes_{p=1}^{P} \big((i_p-1)\Delta x_p, i_p \Delta x_p\big).$$

Also define

$$W_p(i_1,\dots,i_P) := w_p\left(\left(i_1-\frac{1}{2}\right)\Delta x_1,\dots,i_p\Delta x_p,\dots,\left(i_P-\frac{1}{2}\right)\Delta x_P\right),$$
$$V_p(i_1,\dots,i_P) := v_p\left(\left(i_1-\frac{1}{2}\right)\Delta x_1,\dots,i_p\Delta x_p,\dots,\left(i_P-\frac{1}{2}\right)\Delta x_P\right),$$

where the w_d's and v_d's are those defined in Proposition 6.2.1. In previous chapters we used ω_d and ν_d to denote the values of the coefficient functions on the boundaries of the control volumes. However, to avoid confusing the convection velocities with the relative population sizes we now denote the diffusion coefficient and convection velocities on the boundaries of control volumes by W_p and V_p respectively. The coefficient tensors that impose zero flux boundary conditions are given by

$$\widehat{\boldsymbol{\mathcal{U}}}_p(i_1,\dots,i_P) := \begin{cases} 0 & \text{if } i_p = 1, \\ \frac{W_p(i_p-1)}{\Delta x_p^2} - \frac{\min\{0,V_p(i_p-1)\}}{\Delta x_p} & \text{otherwise.} \end{cases}$$
$$\widehat{\boldsymbol{\mathcal{L}}}_p(i_1,\dots,i_P) := \begin{cases} 0 & \text{if } i_p = I_p, \\ \frac{W_p(i_p)}{\Delta x_p^2} + \frac{\max\{0,V_p(i_p)\}}{\Delta x_p} & \text{otherwise.} \end{cases}$$

$$\widehat{\boldsymbol{\mathcal{C}}}_p(i_1,\dots,i_P) := \begin{cases} -\frac{W_p(i_p)}{\Delta x_p^2} - \frac{\max\{0,V_d(i_d)\}}{\Delta x_p} & \text{if } i_p = 1, \\ -\frac{W_p(i_p)+W_p(i_p-1)}{\Delta x_p^2} - \frac{\max\{0,V_p(i_p)\}}{\Delta x_p} + \frac{\min\{0,V_p(i_p-1)\}}{\Delta x_p} & \text{if } 1 < i_p < I_p, \\ -\frac{W_p(i_p-1)}{\Delta x_p^2} + \frac{\min\{0,V_p(i_p-1)\}}{\Delta x_p} & \text{if } i_p = I_p, \end{cases}$$

and the tensor $\boldsymbol{\mathcal{C}}$ is defined by

$$\boldsymbol{\mathcal{C}} := \sum_{p=1}^{P} \widehat{\boldsymbol{\mathcal{C}}}_p.$$

To reflect the fact that we are working with rescaled time the positivity preserving time step size given by Algorithm 3 will be denoted by $\Delta\tau$ and the number of time steps by

N_τ. Once a TT approximation $\overline{\boldsymbol{\Phi}}^{N_\tau}$ of ϕ is found, midpoint approximations can be used to compute the integral in (6.2) to approximate the AFS. First, define a $2P$-dimensional time homogeneous function

$$B(d_1, x_1, d_2, x_2, \ldots, d_P, x_P) := \prod_{p=1}^{P} \binom{N_p}{d_p} x_p^{d_p} (1 - x_p)^{N_p - d_p}, \tag{6.7}$$

we call B the *binomial function* (it is a version of the *Binomial distribution function*). We alternate between d_p and x_p in the variables as this groups 2-dimensional factor functions together, which will be useful soon. Entries of the AFS are given by

$$\boldsymbol{\mathcal{M}}(N_\tau \Delta\tau, d_1, d_2 \ldots, d_P) = \int_{[0,1]^P} B(d_1, x_1, d_2, x_2, \ldots, d_P, x_P) \phi(\tau, x_1, x_2 \ldots, x_P) \, \mathrm{d}V(\boldsymbol{x}).$$

For each $(d_1, \ldots, d_P)$ take a midpoint approximation for the binomial function in the control volume $\Omega(i_1, i_2, \ldots, i_P)$;

$$\boldsymbol{\mathcal{B}}(d_1, i_1, d_2, i_2, \ldots, d_P, i_P) := \prod_{p=1}^{P} \binom{N_p}{d_p} \big[(i_p - 1/2)\Delta x_p\big]^{d_p} \big[1 - (i_p - 1/2)\Delta x_p\big]^{N_p - d_p}. \tag{6.8}$$

View this midpoint approximation as an order $2P$ tensor and for each p the mode indexed by d_p has size $N_p + 1$ and the mode indexed by i_p has size I_p. We call $\boldsymbol{\mathcal{B}}$ the *binomial tensor*. In terms of the tensor approximation $\overline{\boldsymbol{\Phi}}^{N_\tau}$ and the binomial tensor the entries of expected AFS may be approximated by

$$\boldsymbol{\mathcal{M}}(N_\tau \Delta\tau, d_1, d_2 \ldots, d_P) = \Delta\mathcal{D} \sum_{i_1, i_2, \ldots, i_P = 1}^{I_1, I_2, \ldots, I_P} \overline{\boldsymbol{\Phi}}^{N_\tau}(i_1, i_2, \ldots, i_P) \boldsymbol{\mathcal{B}}(d_1, i_1, d_2, i_2, \ldots, d_P, i_P), \tag{6.9}$$

where

$$\Delta D := \prod_{p=1}^{P} \Delta x_p.$$

In order to tidy up equations adopt the notation

$$\boldsymbol{\mathcal{M}}^{N_\tau}(d_1, d_2 \ldots, d_P) := \boldsymbol{\mathcal{M}}(N_\tau \Delta\tau, d_1, d_2 \ldots, d_P)$$

for all $(d_1, d_2, \ldots, d_P)$. It is possible to simplify the sum in (6.9) by using n-mode products and an enlarged version of $\overline{\boldsymbol{\Phi}}^{N_\tau}$;

$$\widehat{\boldsymbol{\Phi}}^{N_\tau}(d_1, i_1, d_2, i_2, \ldots, d_P, i_P) := \overline{\boldsymbol{\Phi}}^{N_\tau}(i_1, i_2, \ldots, i_P) \tag{6.10}$$

for all $(d_1, i_1, \ldots, d_P, i_P)$. So, (6.9) becomes

$$\boldsymbol{\mathcal{M}}^{N_\tau}(d_1, d_2 \ldots, d_P) = \Delta\mathcal{D} \sum_{i_1, i_2, \ldots, i_P = 1}^{I_1, I_2, \ldots, I_P} \widehat{\boldsymbol{\Phi}}^{N_\tau}(d_1, i_1, d_2, i_2, \ldots, d_P, i_P).$$

In terms of n-mode products $\boldsymbol{\mathcal{M}}^{N_\tau}$ may be written as

$$\boldsymbol{\mathcal{M}}^{N_\tau} = \Delta\mathcal{D}\big[\widehat{\boldsymbol{\Phi}}^{N_\tau} * \boldsymbol{\mathcal{B}}\big] \times_2 \mathbf{1}_{I_1} \times_4 \mathbf{1}_{I_2} \ldots \times_{2P} \mathbf{1}_{I_P}, \tag{6.11}$$

where $\mathbf{1}_{I_p}$ is a row vector of length I_p with all entries equal to 1.

Computing the Hadamard product requires at most $\mathcal{O}(PJR^4)$ operations, where

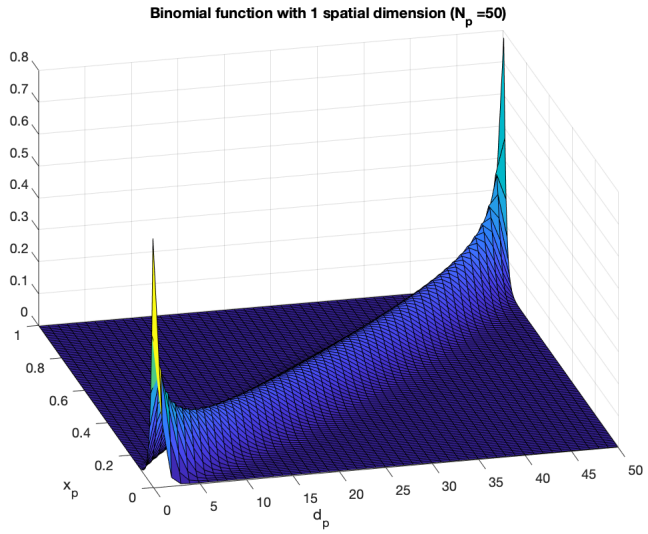
$$J := \max\{N_1 + 1, I_1, N_2 + 1, I_2, \ldots, N_P + 1, I_P\}.$$

Computing each individual n-mode product requires at most $\mathcal{O}(NIR^2)$ operations, where N is the largest population size. There are a total of P n-mode products to compute, so computing all n-mode products requires at most $\mathcal{O}(PNIR^2)$ operations. The scalar multiplication by $\Delta\mathcal{D}$ will require at most $\mathcal{O}(JR^2)$ operations. Therefore, the total cost of computing the AFS in TT format if $\widehat{\boldsymbol{\Phi}}^{N_\tau}$ and $\boldsymbol{\mathcal{B}}$ are in TT format is

$$\mathcal{O}(PJR^4 + PNIR^2).$$

If the binomial tensor $\boldsymbol{\mathcal{B}}$ and the enlarged tensor $\widehat{\boldsymbol{\Phi}}^{N_\tau}$ are known in TT format (6.11) gives an efficient method to approximate the AFS. However, the TT-FVM algorithm doesn't return $\widehat{\boldsymbol{\Phi}}^{N_\tau}$ and cannot be altered to do so without sacrificing computational efficiency. Creating $\widehat{\boldsymbol{\Phi}}^{N_\tau}$ from $\overline{\boldsymbol{\Phi}}^{N_\tau}$ in the TT format is easily done by adding non-essential dimensions. A non-essential dimension is a mode or dimension along which the tensor is constant, such as those indexed by the d_p's in $\widehat{\boldsymbol{\Phi}}^{N_\tau}$. The TT-toolbox contains a function, `add_non_essential_dims`, which does this with negligible computational cost.

Figure 6.1: 2-dimensional binomial function



A more difficult task is to create a TT approximation of the binomial tensor. One would think that the midpoint approximation (6.8) gives everything needed to compute a TT-cross approximation. However, due to the successive multiplications in the definition of the binomial function, the binomial tensor has entries that are very close to zero, while others are significantly larger than zero. Figure 6.1 shows a 2-dimensional binomial function with one spatial dimension. Notice the ridge along the diagonal. Successively multiplying 2-dimensional binomial functions to create the binomial tensor will exaggerate this ridge along the diagonal. TT-cross approximations do not deal well

with such sharp ridges. The approximations that the TT-cross algorithm produces will be inaccurate with large TT ranks. However, the structure of the binomial tensor can be taken advantage of. For each $p \in \{1, \dots, P\}$ define

$$\boldsymbol{B}_p(d_p, i_p) := \binom{N_p}{d_p} \big[(i_p - 1/2)\Delta x_p\big]^{d_p} \big[1 - (i_p - 1/2)\Delta x_p\big]^{N_p - d_p}. \tag{6.12}$$

The binomial tensor can also be written as

$$\boldsymbol{\mathcal{B}}(d_1, i_1, d_2, i_2, \dots, d_P, i_P) = \boldsymbol{B}_1(d_1, i_1)\boldsymbol{B}_2(d_2, i_2) \dots \boldsymbol{B}_P(d_P, i_P). \tag{6.13}$$

Since the matrices $\boldsymbol{B}_p$ will not have such an exaggerated ridge on the diagonal it is possible to approximate them more accurately by a skeleton-decomposition. So suppose that for each p the matrix $\boldsymbol{B}_p$ has rank R_p and there exists matrices $\boldsymbol{U}_p \in \mathbb{R}^{(N_p+1)\times R_p}$ and $\boldsymbol{V} \in \mathbb{R}^{R_p \times I_p}$ such that

$$\boldsymbol{B}_p(d_p, i_p) = \boldsymbol{U}_p(d_p, :)\boldsymbol{V}_p(:, i_p).$$

In index form the binomial tensor may then be written as

$$\boldsymbol{\mathcal{B}}(d_1, i_1, d_2, i_2, \dots, d_P, i_P) = \boldsymbol{U}_1(d_1, :)\boldsymbol{V}_1(:, i_1)\boldsymbol{U}_2(d_2, :)\boldsymbol{V}_2(:, i_2) \dots \boldsymbol{U}_P(d_P, :)\boldsymbol{V}_P(:, i_P),$$

which immediately gives a TT decomposition of $\boldsymbol{\mathcal{B}}$. The cores of the decomposition are the third-order tensors $\boldsymbol{\mathcal{U}}_p \in \mathbb{R}^{1\times(N_p+1)\times R_p}$ and $\boldsymbol{\mathcal{V}}_p \in \mathbb{R}^{R_p \times I_p \times 1}$ given by

$$\boldsymbol{\mathcal{U}}_p(:, d_p, :) = \boldsymbol{U}_p(d_p, :), \quad \text{and} \quad \boldsymbol{\mathcal{V}}_p(:, i_p, :) = \boldsymbol{V}_p(:, i_p).$$

Evaluating one entry in the matrix $\boldsymbol{B}_p$ requires $\mathcal{O}(N_p)$ operations, due to the binomial coefficient. Computing a skeleton decomposition of $\boldsymbol{B}_p$ with a maxvol algorithm requires $\mathcal{O}(J_p R^2)$ matrix entry evaluations and further $\mathcal{O}(J_p R^3)$ operations, where $J_p = \max(N_p, I_p)$. There are P of these skeleton decompositions that need to be found, so the total cost of approximating the binomial tensor is

$$\mathcal{O}(PNJR^2 + PJR^3), \tag{6.14}$$

where $J := \max\{N_1, I_1, N_2, I_2, \dots, N_p, I_p\}$ and $N := \max\{N_1, N_2 \dots, N_p\}$. The logic from the product (6.13) up to this point actually proves a more general property of the TT decomposition and gives an algorithm to create a TT decomposition for a certain class of tensors. This is summarised in Proposition 6.2.3 and formal pseudo-code for creating the approximation is given in Algorithm 6.

Remark 6.2.2
When implementing Algorithm 6 in Matlab to compute the binomial tensor the built-in command `binopdf` was not used to evaluate the entries in the skeleton decomposition. Instead, a hard coded version of the binomial coefficient was created using the `factorial` command. This hard coded binomial coefficient function was then used to create a hard coded version of the binomial distribution function. Evaluating entries in this way appeared to be significantly faster than using the built in `binopdf` command. The built in `nchoosek` command for the binomial coefficient was not used either, because it is not properly vectorised in the version of Matlab [36] that was used. For this reason, it did not work well with commands from the TT-toolbox used to form cross approximations.

Proposition 6.2.3
Suppose that $\boldsymbol{\mathcal{X}}$ is an order $2P$ tensor with mode sizes $J_1, I_1, J_2, I_2, \ldots, J_P, I_P$. If for each $p \in \{1, \ldots, P\}$ there exists a rank R_p matrix $\boldsymbol{X}_p \in \mathbb{R}^{J_p \times I_p}$ such that

$$\boldsymbol{\mathcal{X}}(j_1, i_1, j_2, i_2, \ldots, j_p, i_p) = \boldsymbol{X}_1(j_1, i_1)\boldsymbol{X}_2(j_2, i_2)\ldots\boldsymbol{X}_P(j_P, i_P), \tag{6.15}$$

Then there exists a TT decomposition for $\boldsymbol{\mathcal{X}}$ with TT ranks R'_k given by

$$R'_k = \begin{cases} 1 & \text{if } k \text{ is odd,} \\ R_p & \text{if } k \text{ is even and } k = 2p, \end{cases} \qquad \forall k \in \{1, \ldots, 2P\}.$$

Algorithm 6 requires at most

$$\mathcal{O}\big(PF_xJR^2 + PJR^3\big)$$

operations to create this decomposition, where $J = \max\{J_1, I_1, J_2, I_2, \ldots, J_P, I_P\}$ and F_x is the maximum computational cost of evaluating an entry of $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_P$.

Algorithm 6: TT approximation of tensors with separable indices

---

Result: $\boldsymbol{\mathcal{X}}$ in TT format with cores $\boldsymbol{\mathcal{U}}_1, \boldsymbol{\mathcal{V}}_1, \ldots, \boldsymbol{\mathcal{U}}_P, \boldsymbol{\mathcal{V}}_P$.
Input:
2-dimensional factor functions $\boldsymbol{X}_p$ such that

$$\boldsymbol{\mathcal{X}}(j_1, i_1, j_2, i_2, \ldots, j_p, i_p) = \boldsymbol{X}_1(j_1, i_1)\boldsymbol{X}_2(j_2, i_2)\ldots\boldsymbol{X}_P(j_P, i_P)$$

.
Main Algorithm:
for $p = 1, ..., P$ **do**
  $[\boldsymbol{\mathcal{U}}_p, \boldsymbol{\mathcal{V}}_p] = \text{maxvol}(\boldsymbol{X}_p)$.
end
Return $\boldsymbol{\mathcal{X}}$ in TT format with cores $\boldsymbol{\mathcal{U}}_1, \boldsymbol{\mathcal{V}}_1, \ldots, \boldsymbol{\mathcal{U}}_P, \boldsymbol{\mathcal{V}}_P$

With Algorithm 6 we now have everything that we need to approximate the allele frequency spectrum at time $\tau = \mathcal{T}$ given the proper parameters. The full process is given as pseudo code in Algorithm 7 and its computational cost is summarised in Proposition 6.2.4 below.

Proposition 6.2.4
Suppose we are given the model parameters: population sizes $N_1, N_2, \ldots N_P$, relative population sizes $\nu_1, \nu_2, \ldots, \nu_P$, scaled migration rates $M_{p \leftarrow q}$ for each $p \in \{1, \ldots, P\}$ and $q \neq p$, initial density ϕ_0 and a final time $\mathcal{T}$. After choosing numerical parameters $I_1, I_2, \ldots, I_P$ denoting the number of control volumes along the $x_1, x_2, \ldots, x_P$ axis respectively, Algorithm 7 returns an approximation of the expected AFS $\boldsymbol{\mathcal{M}}^{N_\tau}$ in no more than

$$\mathcal{O}\big(N_\tau P^4 I\mathcal{R}_0^3 + N_\tau P^2 I\mathcal{R}_0^2 R^2 + \varphi_c PI\mathcal{R}_0^2 + P^3 IR^2 + PNJR^2 + PJR^3\big) \tag{6.16}$$

operations.

Proof:
Recall from Theorem 5.1.3 that the total computational complexity of the TT-FVM algorithm is

$$\mathcal{O}\big(N_\tau P^4 I\mathcal{R}_0^3 + N_\tau P^2 I\mathcal{R}_0^2 R^2 + \varphi_c P I\mathcal{R}_0^2 + (W_c + V_c)P^2 I R^2\big), \tag{6.17}$$

where W_c is the maximum cost of evaluating an entry of the diffusion coefficient, V_c is the maximum cost of evaluating an entry of the convection velocity and φ_c the maximum cost of an evaluation of the initial density ϕ. From equation (6.14) it is known that the total computational complexity of computing the binomial tensor is

$$\mathcal{O}(PNJR^2 + PJR^3) \tag{6.18}$$

where $J = \max\{N_1, I_1, N_2, I_2, \ldots, N_P, I_P\}$. By combining (6.17) and (6.18) an estimate for the computational complexity of Algorithm 7 can be derived.

Recall that entries of the diffusion coefficient are given by

$$w_p(\boldsymbol{x}) = \frac{x_p(1 - x_p)}{2\nu_p},$$

for each $p \in \{1, \ldots, P\}$. The number of operations in this function does not change when model parameters are changed. Therefore, $W_c = \mathcal{O}(1)$. Also recall that the entries of the convection velocity vector are given by

$$v_p(\boldsymbol{x}) := \lambda_p x_p + \sum_{q=1,\ q\neq p} M_{p\leftarrow q} x_q - \frac{1}{2\nu_p}.$$

and

$$\lambda_p := \frac{1}{\nu_p} - \sum_{q=1\ q\neq p}^{P} M_{p\leftarrow q}$$

for each $p \in \{1 \ldots, P\}$. Computing one of the λ_p's requires P operations and there are P of them to compute. So pre-computing the λ_p's has a one time cost of $\mathcal{O}(P^2)$ operations. For all p an evaluation of ν_p requires P multiplications and P additions from which it follows that $V_c = \mathcal{O}(P)$. Computing the TT approximation of the density ϕ at $\tau = \mathcal{T}$ with the TT-FVM algorithm then requires at most

$$\mathcal{O}(N_\tau P^4 I\mathcal{R}_0^3) + \mathcal{O}(N_\tau P^2 I\mathcal{R}_0^2 R^2) + \mathcal{O}(\varphi_c P I\mathcal{R}_0^2) + \mathcal{O}(P^3 I R^2), \tag{6.19}$$

operations. Combining (6.19) with (6.18) gives the result.

□

The growth of N_τ is very dependent on the model parameters which will be shown in Section 6.3.3. So, for simplicity it is omitted from the estimate (6.16).

Algorithm 7: Compute AFS in TT-format

Result: The AFS $\boldsymbol{\mathcal{M}}$ at time $\tau = \mathcal{T}$ in TT format.

Input:

Model Parameters:

Initial density ϕ_0.

Target time $\mathcal{T}$.

Population size N_p, relative population size ν_p and scaled migration rates $M_{p \leftarrow q}$ for all $p = 1, \ldots, P$.

FVM parameters:

Number of control volumes along each dimension $I_1, \ldots, I_P$.

Initialisation:

Set $\Delta x_p = I_p^{-1}$ for all $p = 1, \ldots, P$.

Compute $\Delta\mathcal{D} = \prod_{p=1}^{P} \Delta x_p$.

Let $\mathbf{1}_{I_p}$ be a row vector of length I_p with all entries equal to 1, for all p.

Main Algorithm:

1 $\left[\overline{\boldsymbol{\Phi}}^{N_\tau}\right] = \text{TT-FVM}(\phi_0, \mathcal{T}, \Delta x_1, \ldots, \Delta x_p)$.

2 Define the enlarged tensor

$$\widehat{\boldsymbol{\Phi}}^{N_\tau}(d_1, i_1, d_2, i_2, \ldots, d_P, i_P) := \overline{\boldsymbol{\Phi}}^{N_\tau}(i_1, i_2, \ldots, i_P).$$

3 Compute the binomial tensor $\boldsymbol{\mathcal{B}}$ in TT-format with Algorithm 6.

4 Compute AFS in TT-format by

$$\boldsymbol{\mathcal{M}} = \Delta\mathcal{D}\left[\widehat{\boldsymbol{\Phi}}^{N_\tau} * \boldsymbol{\mathcal{B}}\right] \times_2 \mathbf{1}_{I_1} \times_4 \mathbf{1}_{I_2} \ldots \times_{2P} \mathbf{1}_{I_P}.$$

5 **Return**: The AFS, $\boldsymbol{\mathcal{M}}$, in TT-format.

6.3 Numerical Experiments

In this section the various elements of Algorithm 7 are tested. First, the new method for computing a TT approximation of the binomial tensor is tested . Second, Algorithm 7 is tested on a three-dimensional example with various grid sizes to investigate convergence and computation time. The third and final test that was performed was to run Algorithm 7 on a constant grid size while the model parameters were varied. This was done to investigate the effect of model parameters on the numerical method as a whole.

6.3.1 Approximating the Binomial Tensor

To test Algorithm 6 as method for generating the binomial tensor three parameters were varied: the number of control volumes; population sizes; and the number of populations. For the purposes of testing, it is assumed that all populations are the same size.

To test the effects of an increase in the number control volumes the number of populations was fixed at 2 and the population sizes at 50. These restrictions allow for the computation of the full binomial tensor in reasonable time. An equal number of control volumes I_Ω along both spatial dimensions was also assumed, and I_Ω was varied from 25 to 200 at increments of 25. More than 200 control volumes was not possible because the machine that was used ran out of memory while generating the full binomial tensor. Table 6.1 shows how a normal cross approximation (we call this the naive approach) compares to our new method based on Algorithm 6. The values in Table 6.1 are rounded to 4 decimal places. Computation times, relative errors and maximum TT ranks were all measured (The relative error is the ratio between the error in low rank approximation and the Frobenius norm of the full array). Error was measured in the Frobenius norm.

Table 6.1: Comparison of methods for computing the Binomial Tensor

	Computation time (s)		Relative error		Maximum TT-rank	
I_Ω	Naive	New	Naive	New	Naive	New
25	1.9618	0.2233	0.0516	3.5058×10^{-15}	16	25
50	1.8243	0.3560	0.4456	3.0543×10^{-12}	14	41
75	1.3476	0.7169	0.6924	1.0763×10^{-11}	10	43
100	1.6925	0.880	0.6938	9.5413×10^{-12}	10	44
125	1.4925	0.9550	0.7958	4.0100×10^{-12}	8	45
150	3.5096	1.0603	0.4013	5.5850×10^{-12}	12	45
175	4.8185	1.5815	0.4306	6.8004×10^{-12}	14	45
200	3.2639	1.4733	0.6086	7.7722×10^{-12}	10	45

For this test it is seen from the first two columns of Table 6.1 that the new method was slightly faster than the naive approach. The new method took at most 1.5815 seconds while the naive method took at least 1.3476 seconds. On average the naive method took took 3.4789 times longer than the new method. One can see from the third and fourth columns of Table 6.1 that the new method is significantly more accurate than the naive method. On average our method has a relative accuracy 12 orders of magnitude smaller

than that of the naive method. This massive increase in accuracy accompanied by a slight decrease in computation time makes our new method very attractive. However, one should note that the result of the new method has much higher TT ranks. In some applications this can be a problem. In order to see how much the increased TT ranks affect computation time we also used the binomial tensors that were generated in this test to compute the AFS. The initial condition ϕ_0 was assumed to be a normal distribution centred at $(0.5, 0.5)$ with standard deviation 0.1 in both variables. Computation times, relative errors and maximum TT ranks were measured again. (The relative error is the ratio between the error in low rank approximation and the Frobenius norm of the full array). The results are shown in Table 6.2, all values shown there are rounded to 4 decimal places.

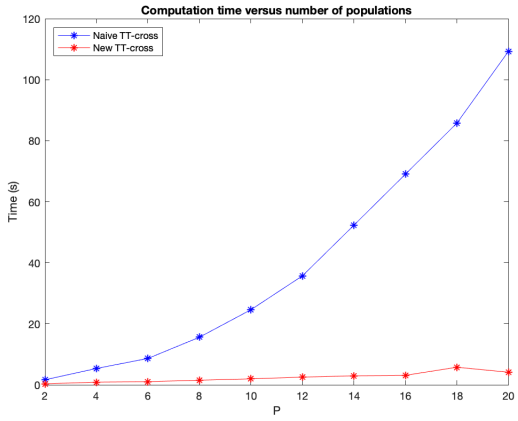
Table 6.2: AFS computation with naive and new binomial tensors

	Computation time (s)		Relative error		Maximum TT-rank	
I_Ω	Naive	New	Naive	New	Naive	New
25	0.0024	0.0033	0.2220	3.005×10^{-15}	16	25
50	0.0020	0.0018	0.0189	3.0171×10^{-15}	14	41
75	0.0021	0.0020	0.5989	3.9016×10^{-15}	10	43
100	0.0019	0.0022	0.0486	4.8104×10^{-15}	10	44
125	0.0028	0.0033	0.3062	8.8387×10^{-14}	8	45
150	0.0079	0.0138	0.0271	1.6666×10^{-14}	12	45
175	0.0085	0.0078	0.0371	1.1021×10^{-13}	14	45
200	0.0079	0.0114	0.1762	2.3140×10^{-13}	10	45

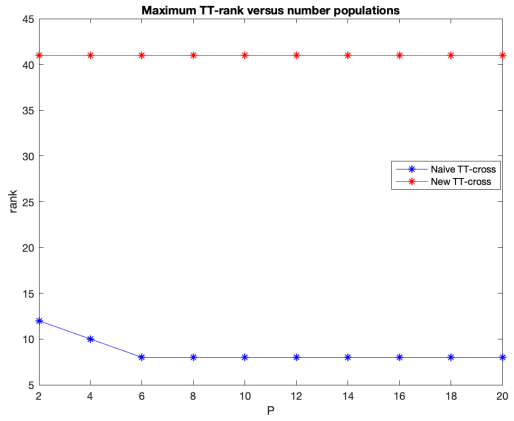
On average computing the AFS using a binomial tensor resulting from our method was 0.8699 times as fast as computing the AFS using a binomial tensor resulting from a naive TT-cross approximation. The TT-ranks of the resulting AFS tensors had the same bounds as that of the binomial tensors. The relative errors for both AFS's are less than the relative error in the binomial tensors, still using a binomial tensor from our method gave a much more accurate approximation. So, in this case the new method provides a very significant increase in accuracy with a decrease in computation time. One should keep in mind that the ranks of the resultant AFS tensor are not small, but a lower rank approximation can always be found with the TT-rounding algorithm.

Next the effect that increasing the number of populations has on the method was investigated. In this case the population sizes were fixed at 50 and the number of control volumes along each dimension to 50. TT approximations of the binomial tensor were created for the number of populations $P \in \{2, 4, 6 \dots 20\}$. Figure 6.2 shows the computation times for these approximations. Again, it was observed that the new method was faster than the naive method. In fact, the new method scaled nearly linearly with the number of populations, as predicted by Proposition 6.2.3, while the naive method scaled nearly quadratically. Even though the new method was faster, it still gave a resultant tensor with higher TT ranks. Surprisingly, both methods had almost no increase in TT-ranks as dimension increased, in fact the new method always returned a TT tensor with a maximum rank of 41. The range of P was limited to 20 because the naive approach fails at $P = 21$ and beyond. It failed in the sense that it

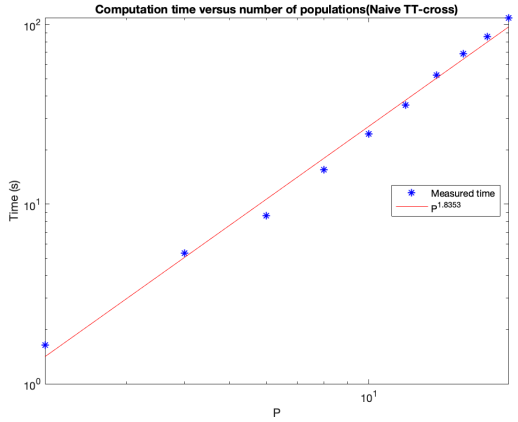
Figure 6.2: Effects of increasing the number of populations



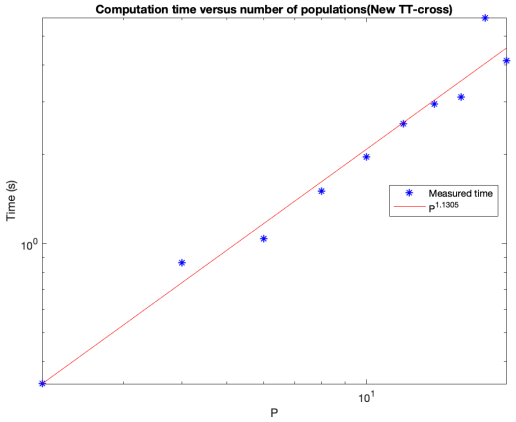
(a) Computation times.



(b) Maximum TT-ranks.



(c) Log-log plot of computation time for the naive method

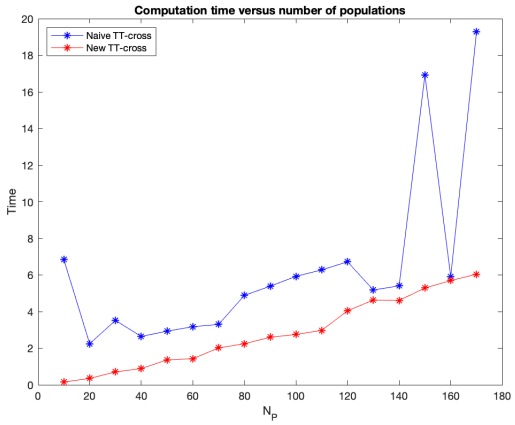


(d) Log-log plot of computation time for the new method.

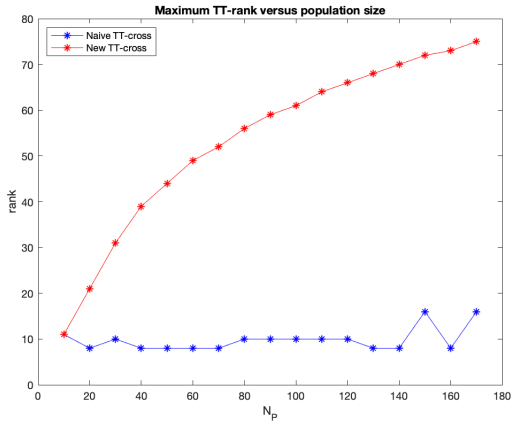
would always return the zero tensor because the entries of the binomial tensor are too small (below machine precision) and the "ridge" along the diagonal is too sharp. In practice we have not yet found a limit to the number of populations that the new method can handle. At $P = 100$ it was still able to run and only took 19.05 seconds. Even if the naive method could approximate the binomial tensor with $P = 100$, the fact that its computation time scale quadratically with P means that it would take about 45 minutes. The new method does not fail at $P = 100$, because it only ever needs to compute one binomial distribution function at a time and never actually needs to compute the product of the distributions to form an approximation. So, in theory the new method should be able to handle an arbitrary number of populations.

Finally, the effect of increasing population sizes was considered. The number of populations was fixed at 3 and the number of control volumes along each dimension at 100. It was also assumed that all population sizes were equal to N_P. Both a naive TT cross and the new method were ran for $N_P \in \{10, 20, 30, \ldots, 170\}$. Figure 6.3 shows the results. The computation time of the new method grew much more smoothly than that of the naive method. Furthermore, the computation time of the new method grew as $\mathcal{O}(N_P^{1.3016})$ while the computation time of the naive method followed no real pattern.

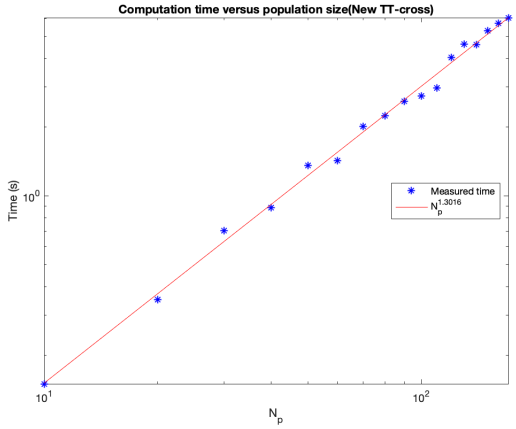
Figure 6.3: Effects of increasing the population sizes



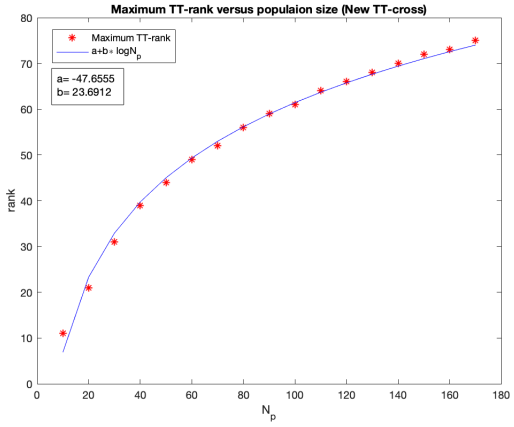
(a) Computation time.



(b) Maximum TT-ranks



(c) log-log plot of computation time (New TT-cross).



(d) Maximum TT-rank grows logarithmically.

Comparing Figure 6.3a and Figure 6.3b that the spikes in the computation time of the naive approach were accompanied by spikes in the maximum TT-ranks . While the TT-ranks of the approximations from the new method were higher again, something interesting happened. Figure 6.3c shows that the maximum TT rank of approximations formed using the new method grew logarithmically with population size. The logarithm in Figure-6.3c was fitted by minimising the squared error. So far, this is just an experimental observation. We have not looked at deriving this rank growth from theory. However, if the logarithmic growth holds in general the rank growth will not be detrimental for large population sizes. Checking that the logarithmic rank growth continues for larger population sizes was not possible as both methods failed for population sizes greater than 170. Both methods got stuck in the maxvol parts of their algorithms because all sub-matrices had determinants smaller than machine accuracy. This meant that maximum volume sub-matrices could not be found.

6.3.2 Simulating Three Populations

In this section the impact of increasing the number of control volumes has on Algorithm 7 is investigated. In particular the order of convergence is estimated, and the impact

that increasing the number of control volumes has on computation time is also looked at.

To test Algorithm 7 three populations of equal size and with symmetric migration rates were simulated. In other words, three identical populations were assumed. To choose parameters for the simulation the effective population size of humans N_{ef} was used. In [2] the effective population size of humans is estimated to be between 10,000 and 20,000. The middle of that range was chosen so $N_{ef} = 15,000$. It was also assumed that $N_1 = N_2 = N_3 = 5000$ then

$$\nu_1 = \nu_2 = \nu_3 = \frac{1}{3}.$$

For scaled migration rates it was assumed that

$$\begin{aligned} M_{1\leftarrow 2} &= M_{1\leftarrow 3} = 0.01, \\ M_{2\leftarrow 1} &= M_{2\leftarrow 3} = 0.01, \\ M_{3\leftarrow 1} &= M_{3\leftarrow 2} = 0.01. \end{aligned}$$

This translates to $m_{p\to q} \approx 3.3 \times 10^{-7}$, which means there is some migration between the populations, but for the most part individuals stay in the population in which they were born. Finally, we ran the simulation up to $\tau = 0.02$ which translates to $t = 600$ generations. Assuming a generation length of 30 years, [37], 600 generations correspond to 18,000 years. This is a substantial amount of time to cover in our model.

In this example there is no exact analytic solution with which to compare the approximations in order to investigate convergence. Instead, the *numerical order of convergence* was used as a measure of convergence. Let $\overline{\boldsymbol{\Phi}}_{I_\Omega}$ and $\overline{\boldsymbol{\Phi}}_{2I_\Omega}$ be estimates for ϕ at $\tau = 0.02$ generated by our finite volume method with I_Ω and $2I_\Omega$ control volumes respectively, for some positive integer I_Ω. If there exists positive real numbers η_ϕ and C_ϕ such that

$$||\overline{\boldsymbol{\Phi}}_{I_\Omega} - \overline{\boldsymbol{\Phi}}_{2kI_\Omega}||_{L^2} = C_\phi (I_\Omega)^{-\eta_\phi} + \mathcal{O}(I_\Omega^{-\eta_\phi - 1}) \quad \forall I_\Omega \in \mathbb{Z}^+, \tag{6.20}$$

then η_ϕ is called the *numerical order of convergence* of our numerical method. In the case that (6.20) holds when $\overline{\boldsymbol{\Phi}}_{2kI_\Omega}$ is replaced with the exact solution η_ϕ is called the *order of convergence* of our numerical method. If (6.20) holds then and a positive integer I_Ω is fixed the sequence of approximations $\{\overline{\boldsymbol{\Phi}}_{2^k I_\Omega}\}_{k\in\mathbb{N}}$ is a Cauchy sequence in L^2. Therefore, the sequence will converge as k goes to infinity.

Given three approximations $\overline{\boldsymbol{\Phi}}_{I_\Omega}$, $\overline{\boldsymbol{\Phi}}_{2I_\Omega}$ and $\overline{\boldsymbol{\Phi}}_{4I_\Omega}$ it follows from (6.20) that

$$\begin{aligned} \frac{||\overline{\boldsymbol{\Phi}}_{I_\Omega} - \overline{\boldsymbol{\Phi}}_{2I_\Omega}||_{L^2}}{||\overline{\boldsymbol{\Phi}}_{2I_\Omega} - \overline{\boldsymbol{\Phi}}_{4I_\Omega}||_{L^2}} &= \frac{C_\phi (I_\Omega)^{-\eta_\phi} + \mathcal{O}(I_\Omega^{-\eta_\phi - 1})}{C_\phi (2I_\Omega)^{-\eta_\phi} + \mathcal{O}(I_\Omega^{-\eta_\phi - 1})} \\ &= \frac{1 + \mathcal{O}(I_\Omega^{-1})}{2^{-\eta_\phi} + \mathcal{O}(I_\Omega^{-1})} \\ &= 2^{\eta_\phi} + \mathcal{O}(I_\Omega^{-1}). \end{aligned} \tag{6.21}$$

Dropping the $\mathcal{O}(I_\Omega^{-1})$ term and taking the natural logarithm of both sides (6.21) the numerical order of convergence may be approximated by

$$\eta_\phi = \frac{\log\left(||\overline{\boldsymbol{\Phi}}_{I_\Omega} - \overline{\boldsymbol{\Phi}}_{2I_\Omega}||_{L^2}\right) - \log\left(||\overline{\boldsymbol{\Phi}}_{2I_\Omega} - \overline{\boldsymbol{\Phi}}_{4I_\Omega}||_{L^2}\right)}{\log(2)}. \tag{6.22}$$

For any given I_Ω the η_ϕ that is computed by (6.22) is only an approximation of the order of convergence, because the $\mathcal{O}(I_\Omega^{-1})$ term in (6.21) was dropped. Therefore, to approximate the numerical order of convergence multiple approximations $\overline{\boldsymbol{\Phi}}_{I_\Omega}$ for increasing I_Ω need to be used. It should be noted that the approximations $\overline{\boldsymbol{\Phi}}_{I_\Omega}$ and $\overline{\boldsymbol{\Phi}}_{2I_\Omega}$, when viewed as tensors, will not have the same number of elements. This means that it is not possible compute their difference entry wise. However, the fact that $\overline{\boldsymbol{\Phi}}_{I_\Omega}$ and $\overline{\boldsymbol{\Phi}}_{2I_\Omega}$ are finite volume approximations of the same function can be used to compute their difference. How this is done is outlined in Appendix A.0.2.

The $\overline{\boldsymbol{\Phi}}_{I_\Omega}$ that were generated with the parameters above were also used to compute the expected AFS for $N_1 = N_2 = N_3 = 100$ in order to check that the estimates of the AFS converge as the number of control volumes I_Ω increase. As with the density ϕ there is no exact solution with which approximations of the AFS can be compared. So, the numerical order of convergence for the AFS was also estimated. Let $\boldsymbol{\mathcal{M}}_{I_\Omega}$ and $\boldsymbol{\mathcal{M}}_{2I_\Omega}$ be estimates for the AFS $\boldsymbol{\mathcal{M}}$ at $\tau = 0.02$ generated by Algorithm 7 with I_Ω and $2I_\Omega$ control volumes respectively, for some positive integer I_Ω. If there exists positive real numbers C_M and η_M such that

$$||\boldsymbol{\mathcal{M}}_{I_\Omega} - \boldsymbol{\mathcal{M}}_{I_\Omega}||_F = C_M(I_\Omega)^{-\eta_M} + \mathcal{O}(I_\Omega^{-\eta_M+1}) \quad \forall I_\Omega \in \mathbb{Z}^+$$

η_M is called the numerical order of convergence of the AFS, and it can be approximated by

$$\eta_M = \frac{\log\left(||\boldsymbol{\mathcal{M}}_{I_\Omega} - \boldsymbol{\mathcal{M}}_{2I_\Omega}||_F\right) - \log\left(||\boldsymbol{\mathcal{M}}_{2I_\Omega} - \boldsymbol{\mathcal{M}}_{4I_\Omega}||_{L^2}\right)}{\log(2)}. \tag{6.23}$$

It should be noted that for both the density approximations $\overline{\boldsymbol{\Phi}}_{I_\omega}$ and the AFS approximations $\boldsymbol{\mathcal{M}}_{I_\Omega}$ the error at each time step is not being measured (this would be called the local truncation error). The error that accumulates at each time step until the final approximation is made is what is measured (this is called the global error). For more on numerical order of convergence see [17].

To estimate for the numerical orders η_ϕ and η_M we ran our model multiple times, each time doubling the number of control volumes. The sequences

$$I_\Omega \in \{20, 40, 80, 160, 320, 640\}$$

and

$$I_\Omega \in \{15, 30, 60, 120, 240, 480\}$$

were used. This gave two sequences of estimates for each numerical order η_ϕ and η_M. Simulations were ran for two different initial values for I_Ω to check that the starting point does not affect the rate of convergence and to get more estimates for the orders of convergence.

The computation times and estimates of the numerical orders of convergence of our simulations are given in Table 6.3. All values shown there are rounded to 4 decimal places. The estimates of both η_ϕ and η_M are relatively stable, but the estimate of η_M varies slightly more than the estimate for η_ϕ. The average estimate $\overline{\eta}_\phi$ for η_ϕ was

$$\overline{\eta}_\phi = 2.4932.$$

The average estimate $\overline{\eta}_M$ for η_M was

$$\overline{\eta}_M = 1.1824.$$

The fact that both of η_ϕ and η_M have relatively consistent estimates is evidence that the approximations of the density ϕ and the AFS $\boldsymbol{\mathcal{M}}$ are converging as the number of control volumes increase. To be sure that the approximations are really converging we would need to run the simulations for more control volumes, however the computation time is a limitation. Figure 6.4 shows how the computation times grew the number of control volumes increased. Figure 6.4a shows that in this case the total computation

Table 6.3: Computation times and orders of convergence

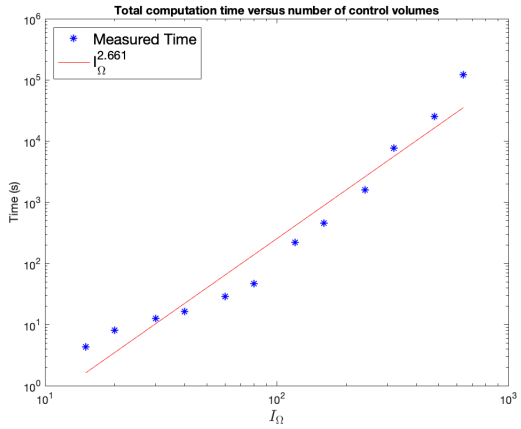
	Computation time (s)			Orders of Convergence	
I_Ω	$\overline{\boldsymbol{\Phi}}_{I_\Omega}$	$\boldsymbol{\mathcal{M}}_{I_\Omega}$	total	η_ϕ	η_M
20	7.3699	0.6243	7.9941	2.4864	1.2639
40	14.7967	1.5702	16.3670	2.4945	1.1651
80	43.2016	3.6452	46.8468	2.4967	1.0236
160	449.5475	7.7031	457.2506	2.5006	1.2272
320	7599.7103	9.8996	7609.6099		
640	121883.2276	17.9209	121901.1485		
	Computation time (s)			Orders of Convergence	
I_Ω	$\overline{\boldsymbol{\Phi}}_{I_\Omega}$	$\boldsymbol{\mathcal{M}}_{I_\Omega}$	total	η_ϕ	η_M
15	3.5867	0.7426	4.3293	2.4796	1.3209
30	11.2340	1.2196	12.4535	2.4925	1.2285
60	25.7368	2.9613	28.6981	2.4954	1.0196
120	214.6734	5.3704	220.0438	2.5000	1.2105
240	1578.0064	12.2370	1590.2434		
480	25435.5297	12.1544	25447.6840		

time grew as $\mathcal{O}(I_\Omega^{2.661})$, and in Figure 6.4c it is seen that the main driver of this super-linear growth is the integration step. The time spent in the integration step of the TT-FVM algorithm grew as $\mathcal{O}(I_\Omega^{3.3974})$. In contrast, the time spent setting the coefficient tensors and computing the AFS both grew approximately linearly with I_Ω.

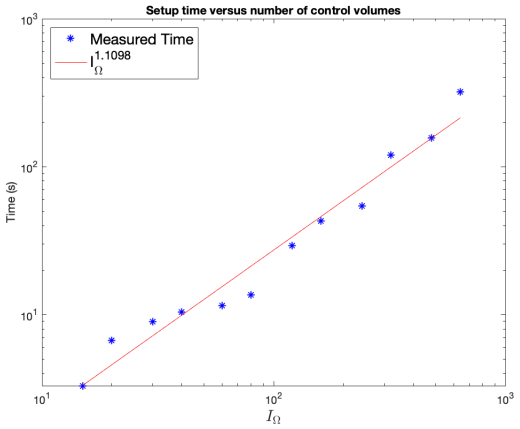
While it is predicted in Proposition 6.2.4 that the total computation time would scale linearly with I_Ω, in that prediction the growth of the TT-ranks or the number of time steps N_τ with the number of control volumes I_Ω was not accounted for. While running our simulations we also kept track of N_τ and the maximum TT-ranks that occurred for the coefficient tensors, and the results are shown in Figure 6.4e and Figure 6.4f. We see that the TT-Ranks did grow as I_Ω, increased but not significantly. Figure 6.4e shows that number of time steps N_τ grew as $\mathcal{O}(I_\Omega^{1.9841})$ that is, the number of time steps grew essentially quadratically with I_Ω. This accounts for the two extra orders of magnitude in the growth of the time spent in the integration step.

In order to speed up our method another step was added to the calculation of the coefficient tensors in Algorithm 3 of Chapter 5. Before starting the integration step the coefficient tensors were rounded to have lower TT ranks. In particular the tensors $\widehat{\boldsymbol{\mathcal{U}}}_p$

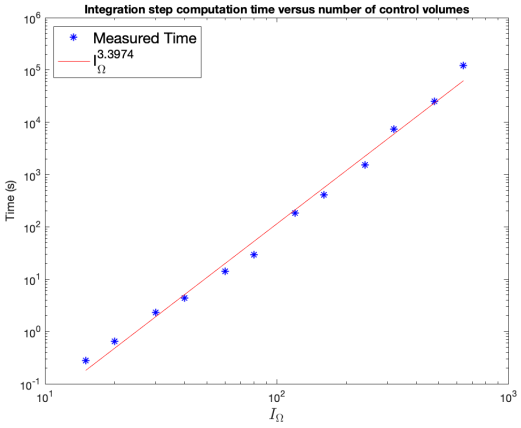
Figure 6.4: Growth of computation times with increasing number of control volumes.



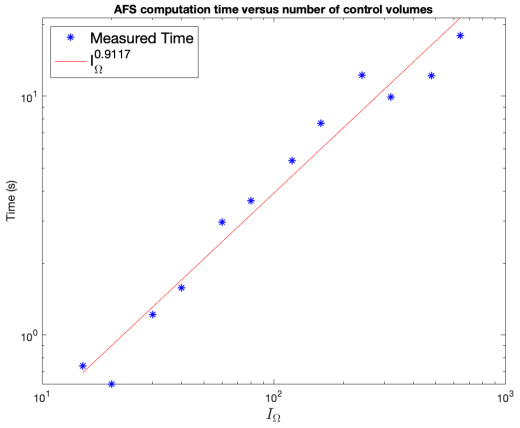
(a) Total computation time.



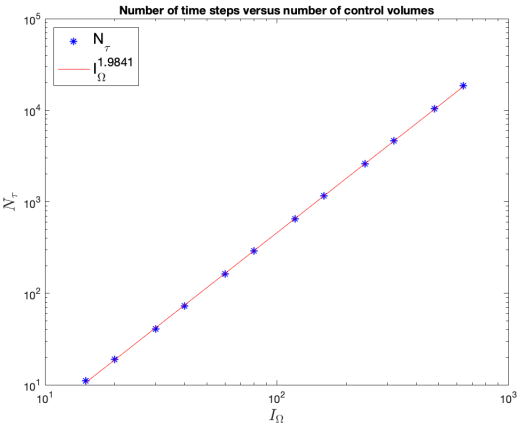
(b) Time to compute coefficient tensors.



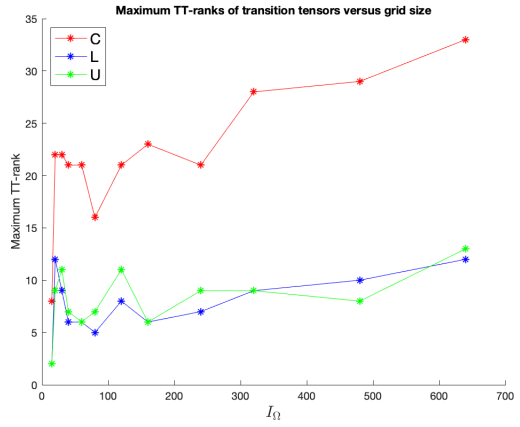
(c) Integration step computation time.



(d) AFS Computation time



(e) Growth of N_τ.



(f) Maximum TT-ranks of the coefficient tensors.

and $\widehat{\boldsymbol{\mathcal{L}}}_p$ were rounded to have TT-ranks no higher than P (which would be 3 in this case). The tensor $\widehat{\boldsymbol{\mathcal{C}}}$, the sum of all $\widehat{\boldsymbol{\mathcal{C}}}_p$'s, was rounded to have TT ranks no higher than $3P$ (which would be 9 in this case).

Each of the roundings will require $\mathcal{O}(PIR^3)$ operations, but since the ranks of the coefficient tensors are known the prediction from Proposition 6.2.4 can be adjusted. Computing the AFS from the initial density ϕ_0 will now require at most

$$\mathcal{O}\big(N_\tau P^4 I\mathcal{R}_0^3 + N_\tau P^4 I\mathcal{R}_0^2 + \varphi_c PI\mathcal{R}_0^2 + P^5 I + PNJR^2 + PJR^3 + PIR^3\big)$$

operations. The references to the rank R that are left are TT ranks that occur in the process of approximating the binomial tensor that are not affected the ranks of the coefficient tensors, or TT-ranks of the coefficient tensors before rounding.

Using the new rounded coefficient tensors, we ran our model again for $I_\Omega \in \{20, 40, 80, 160, 320, 640\}$ and $I_\Omega \in \{15, 30, 60, 120, 240, 480\}$ and estimated the numerical orders of convergence. The results of these simulations are shown in Table 6.4 below. We denote the approximation of the density ϕ at $\tau = 0.02$ by $\widehat{\boldsymbol{\Phi}}_{I_\Omega}$ and the approximation of the AFS by $\widehat{\boldsymbol{\mathcal{M}}}_{I_\Omega}$

Table 6.4: Computation times and orders of convergence, with rounded coefficient tensors

	Computation time (s)			Orders of Convergence	
I_Ω	$\widehat{\boldsymbol{\Phi}}_{I_\Omega}$	$\widehat{\boldsymbol{\mathcal{M}}}_{I_\Omega}$	total	η_ϕ	η_M
20	4.5453	0.6365	5.1818	2.4868	1.2826
40	10.8725	1.5928	12.4653	2.4927	1.0425
80	26.2240	3.8915	30.1155	2.4994	1.2441
160	121.7130	7.6736	129.3866	2.4985	1.0180
320	908.4953	11.1233	919.6187		
640	6581.1167	15.1493	6596.2660		
	Computation time (s)			Orders of Convergence	
I_Ω	$\widehat{\boldsymbol{\Phi}}_{I_\Omega}$	$\widehat{\boldsymbol{\mathcal{M}}}_{I_\Omega}$	total	η_ϕ	η_M
15	1.1389	0.3806	1.5195	2.4797	1.3240
30	5.5416	0.8870	6.4286	2.4919	1.1867
60	16.0518	2.9946	19.0464	2.4961	1.0822
120	59.8680	5.5482	65.4162	2.5004	1.2508
240	364.6335	9.0318	373.6653		
480	2619.9938	13.9256	2633.9194		

In this case the average estimate $\overline{\eta}_\phi$ for η_ϕ was

$$\overline{\eta}_\phi = 2.4932,$$

which is the same as the estimate when the coefficient tensors were not rounded (up to 4 decimal places). The average estimate $\overline{\eta}_M$ for η_M was

$$\overline{\eta}_M = 1.1789,$$

which is very similar to the order of convergence that was estimated when the coefficient tensors were not rounded. The L^2 error between the TT approximations of the density generated by both methods, and the Frobenius error between the AFS's were both measured. Table 6.5 shows the errors and relative errors that were measured.

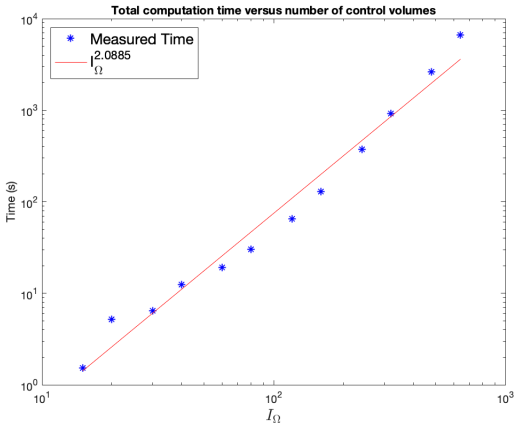
Table 6.5: Comparing estimates with and without rounded coefficient tensors

I_Ω	$\left\|\left\|\overline{\boldsymbol{\Phi}}_{I_\Omega} - \widehat{\boldsymbol{\Phi}}_{I_\Omega}\right\|\right\|_{L^2}$	$\frac{\left\|\left\|\overline{\boldsymbol{\Phi}}_{I_\Omega} - \widehat{\boldsymbol{\Phi}}_{I_\Omega}\right\|\right\|_{L^2}}{\left\|\left\|\overline{\boldsymbol{\Phi}}_{I_\Omega}\right\|\right\|_{L^2}}$	$\left\|\left\|\boldsymbol{\mathcal{M}}_{I_\Omega} - \widehat{\boldsymbol{\mathcal{M}}}_{I_\Omega}\right\|\right\|_F$	$\frac{\left\|\left\|\boldsymbol{\mathcal{M}}_{I_\Omega} - \widehat{\boldsymbol{\mathcal{M}}}_{I_\Omega}\right\|\right\|_F}{\left\|\left\|\boldsymbol{\mathcal{M}}_{I_\Omega}\right\|\right\|_F}$
15	1.2367×10^{-16}	3.1035×10^{-15}	1.3575×10^{-17}	6.2808×10^{-15}
20	2.6173×10^{-6}	1.0032×10^{-4}	1.944×10^{-7}	8.9294×10^{-5}
30	6.9526×10^{-7}	4.8598×10^{-5}	8.9459×10^{-8}	4.0815×10^{-5}
40	2.7295×10^{-7}	2.9274×10^{-5}	5.4832×10^{-8}	2.4940×10^{-5}
60	2.9056×10^{-7}	5.7068×10^{-5}	1.0989×10^{-7}	4.9833×10^{-5}
80	5.1796×10^{-7}	1.5639×10^{-4}	3.1056×10^{-7}	1.4064×10^{-4}
120	1.5977×10^{-7}	8.8500×10^{-5}	1.7420×10^{-7}	7.8786×10^{-5}
160	1.1027×10^{-7}	9.3970×10^{-5}	1.8208×10^{-7}	8.2295×10^{-5}
240	2.5795×10^{-8}	4.0354×10^{-5}	8.0119×10^{-8}	3.6186×10^{-5}
320	2.2973×10^{-9}	5.5315×10^{-5}	1.1213×10^{-7}	5.0627×10^{-5}
480	4.8660×10^{-9}	2.1518×10^{-5}	3.7919×10^{-8}	1.7117×10^{-5}
640	2.6367×10^{-9}	1.7948×10^{-5}	3.1396×10^{-8}	1.4170×10^{-5}

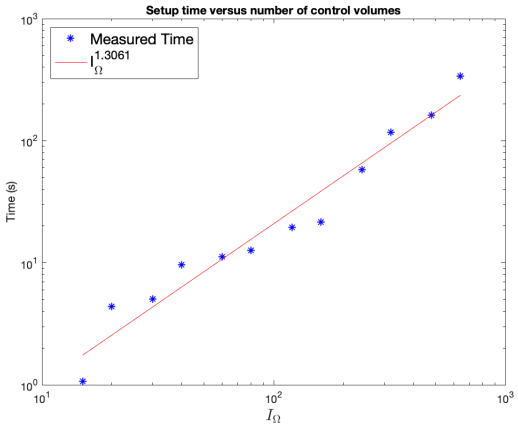
From the errors shown in Table 6.5 we see that after rounding the coefficient tensors we get a reasonable approximation of the density $\overline{\boldsymbol{\Phi}}_{I_\Omega}$ and the AFS $\boldsymbol{\mathcal{M}}_{I_\Omega}$. At the same time a significant decrease in the computation time is also achieved. By comparing the computation times from Table 6.3 and Table 6.5 it is found that on average the time spent computing $\widehat{\boldsymbol{\Phi}}_{I_\Omega}$ with the rounded coefficient tensors was about 5 times less than computing $\overline{\boldsymbol{\Phi}}_{I_\Omega}$ with the unrounded coefficient tensors. At the finest grid, with $I_\Omega = 640$, using the rounded coefficient tensors was about 18 times faster. Figure 6.5 shows how the computation times scaled with the increasing number of control volumes. While Figure 6.5b shows that the setup time grew slightly faster than linearly due to the added rounding procedure, the increase in the setup time was not detrimental. The integration step computation time now grew as $\mathcal{O}(I_\Omega^{3.0523})$, which is essentially cubic growth and slightly better than the $\mathcal{O}(I_\Omega^{3.397})$ growth that was observed when the coefficient tensors were not rounded. The total computation time grew as $\mathcal{O}(I_\Omega^{2.0885})$, which is not significantly better than the $\mathcal{O}(I_\Omega^{2.661})$ growth that was seen when the coefficient tensors were not rounded.

While we did manage to decrease the total computation time by rounding the coefficient tensors, we were not able to significantly improve the scaling laws. This can be explained by the fact that by rounding the coefficient tensors the average time required for one time step to be performed was decreased, but N_τ (the number of time steps required) was not decreased. Figure 6.6a shows how these average times scaled with I_Ω. Unfortunately, it is not possible to decrease N_τ since it is defined by the time step-size $\Delta\tau$, and $\Delta\tau$ is chosen specifically to be the largest time step-size for which our finite volume method is positivity preserving. So, no matter how small the time required for a single time step is, the growth of N_τ will always dominate. This means that our

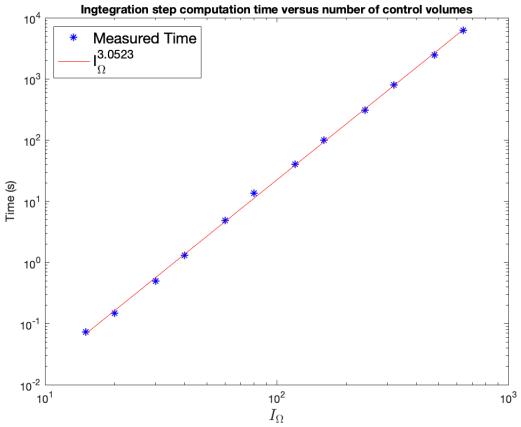
Figure 6.5: Growth of computation times with increasing number of control volumes using rounded coefficient tensors.



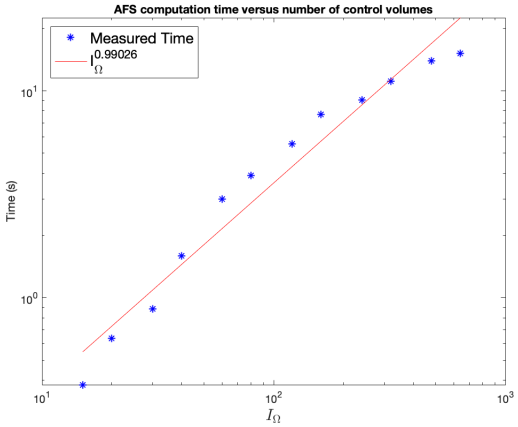
(a) Total computation time.



(b) Time to compute coefficient tensors.



(c) Integration step computation time.



(d) AFS Computation time

methods are not held back by the use of the TT format, but rather by the use of a positivity preserving finite volume method.

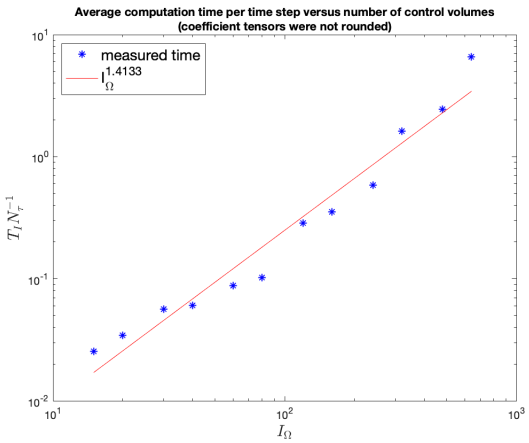
6.3.3 Investigating the Impact of Model Parameters

In Section 6.3.2 Algorithm 7 was tested on one set of model parameters. Here the model parameters are varied to gain a better understanding of how the TT-FVM method is affected by model parameters. It was posited in Section 6.3.2 that the main driver for the increase in computation time of our methods is the number of time steps, N_τ. To gain an understanding of how the model parameters affect N_τ the coefficient tensors were created for different relative population sizes ν_p and scaled migration rates $M_{p\leftarrow q}$.

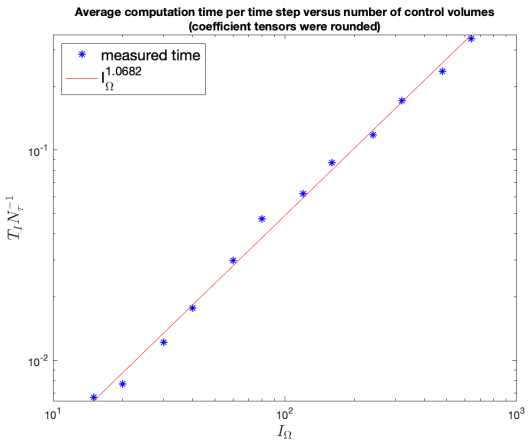
The first test was to fix the migration rates at

$$\begin{aligned} M_{1\leftarrow 2} &= M_{1\leftarrow 3} = 0.01, \\ M_{2\leftarrow 1} &= M_{2\leftarrow 3} = 0.01, \\ M_{3\leftarrow 1} &= M_{3\leftarrow 2} = 0.01, \end{aligned}$$

Figure 6.6: Average Computation time required for one time step
(T_I represents the total time required for the integration step)



(a) Non-rounded coefficient tensors.



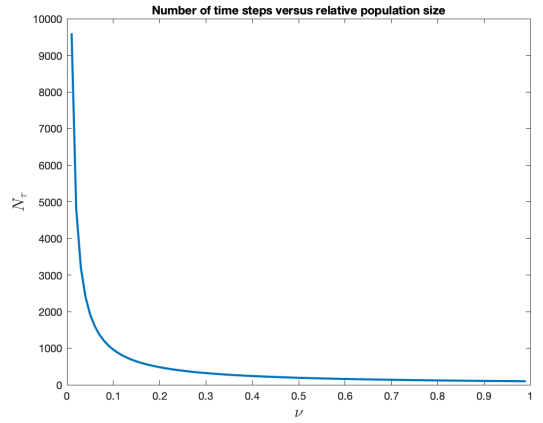
(b) Rounded coefficient tensors

and assume that

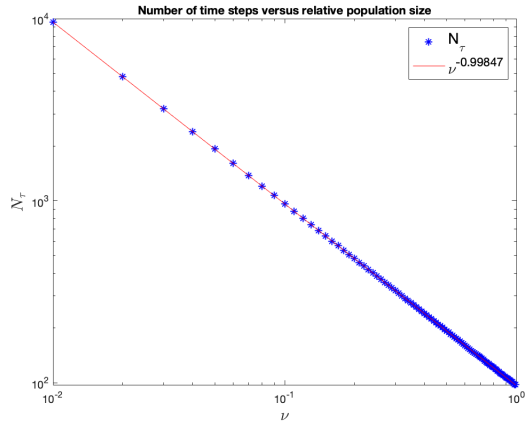
$$\nu_1 = \nu_2 = \nu_3 = \nu.$$

The number of control volumes along each population axis (x_p) was fixed at $I_\Omega = 80$, and ν was varied from 0.1 to 0.9 at intervals of 0.01. The coefficient tensors were generated for each value of ν. At each value of ν the maximum TT ranks of the coefficient tensors and N_τ were recorded. Figure 6.7 shows how N_τ scaled with ν, and in Figure-6.7a it is observed that N_τ scaled as $\mathcal{O}(\nu^{-0.99847})$. The minimum N_τ was 97 which was attained at $\nu = 0.99$, and the maximum N_τ was 9604 which was attained at $\nu = 0.01$. At $\nu = 1/3$ as assumed in Section 6.3.2 we observed $N_\tau = 289$.

Figure 6.7: Growth of N_τ with relative population size



(a) N_τ versus ν.

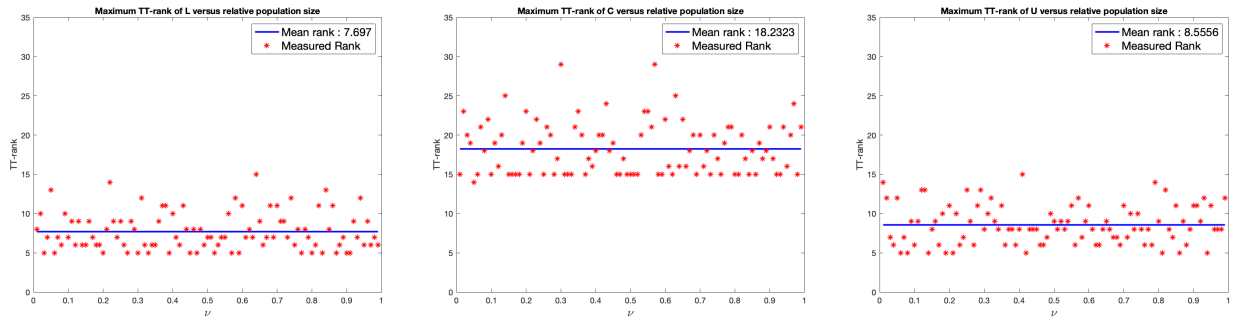


(b) log-log plot of N_τ versus ν.

Th maximum TT ranks of the coefficient tensors before rounding are plotted against ν in Figure 6.10. Notice that the TT ranks were not linked to ν in any meaningful way. The measured TT ranks were essentially random but closely packed around a mean. The randomness is due to the random initial guess used in the `amen_ cross` function

from the TT-toolbox [45] that was used to create TT-cross approximations. This test shows that when decreasing the relative population sizes the total computation time for approximating the AFS can be significantly increased due to the reciprocal growth of N_τ. However, when relative populations are large the computation time can be reduced to an extent.

Figure 6.8: Maximum TT-ranks with increasing relative population sizes



(a) TT-ranks of the $\widehat{\boldsymbol{\mathcal{L}}}_p$'s. (b) TT-ranks of $\boldsymbol{\mathcal{C}}$. (c) TT-ranks of $\widehat{\boldsymbol{\mathcal{U}}}_p$'s.

Next, the relative population sizes were fixed at

$$\nu_1 = \nu_2 = \nu_3 = \frac{1}{3},$$

and symmetric migration was assumed with

$$\begin{aligned} M_{1\leftarrow 2} &= M_{1\leftarrow 3} = m, \\ M_{2\leftarrow 1} &= M_{2\leftarrow 3} = m, \\ M_{3\leftarrow 1} &= M_{3\leftarrow 2} = m. \end{aligned}$$

The number of control volumes was at $I_\Omega = 80$ along each population axis and m was varied from 0.01 to 0.99 at intervals of 0.03. In this case N_τ stayed nearly constant, jumping between 288 and 289 as shown in Figure-6.9. So, when increasing the migration rates the number of time steps is not affected. However, we did find that the TT-ranks of the coefficient tensors are affected.

The main reason why computation time increases when migration rates are increased is the increase in the ranks of the coefficient tensors. In turn, this means that creating a TT-cross approximation of the coefficient tensors and rounding them takes longer. However, it was found that the TT ranks and thus the setup time grew logarithmically with the migration rates (logarithms were fitted by minimising the sum of squared errors). The logarithmic trends were only followed in shape, with some variation from the trend. This variation is caused by the random initial guess in the `amen_cross` function used for cross approximation. From this test we conclude that for larger migration rates the increase in the computation time is due to an increase in the TT ranks. Computation time of the integration step can be improved by rounding the coefficient tensors, but since the TT ranks are high accuracy might be reduced significantly.

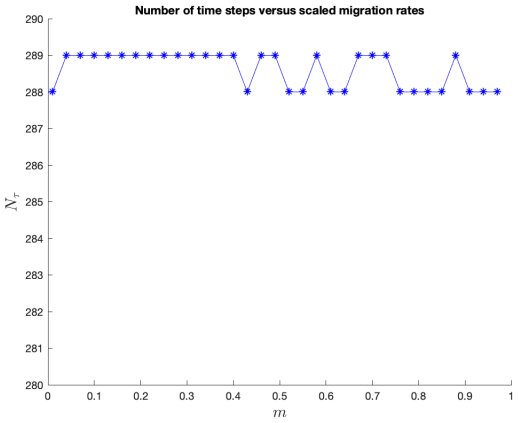


Figure 6.9

To investigate the effect of increasing the number of populations it was assumed that relative population sizes and the scaled migration rates were fixed. It was assumed that

$$\nu_1 = \nu_2 = \nu_3 = \frac{1}{3},$$

and migration was symmetric with

$$\begin{aligned} M_{1\leftarrow 2} &= M_{1\leftarrow 3} = 0.01, \\ M_{2\leftarrow 1} &= M_{2\leftarrow 3} = 0.01, \\ M_{3\leftarrow 1} &= M_{3\leftarrow 2} = 0.01. \end{aligned}$$

The number of populations P was varied from 2 to 16. The number of control volumes along each population axis were fixed at $I_\Omega = 80$. At each value of P, the coefficient tensors were generated. Figure 6.11 shows the results of this test.

For increasing populations it was seen that the maximum TT-ranks of the $\widehat{\boldsymbol{\mathcal{L}}}_p$'s and $\widehat{\boldsymbol{\mathcal{U}}}_p$'s stayed more or less constant at around 18, while the maximum TT-rank of $\boldsymbol{\mathcal{C}}$ grew as $\mathcal{O}(P^{1.4608})$. This is seen in Figure 6.11c and Figure 6.11d. In Figure 6.11b it is seen that N_τ grew approximately linearly with the number of dimensions. The setup time grew as $\mathcal{O}(P^{2.8954})$ which is a result of the increased number of tensors that need to be approximated (which grows linearly with dimension) and the increase in ranks of the tensor $\boldsymbol{\mathcal{C}}$. When increasing the number of populations, the total computation time required to approximate the AFS will increase due to the increase of time steps required and the increase in the ranks of the coefficient tensors.

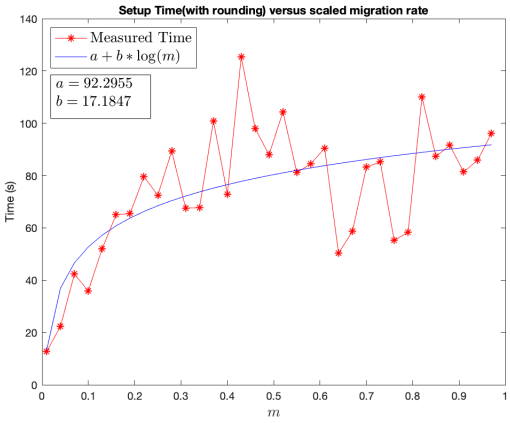
We also examined how the density and the allele frequency spectrum are affected by model parameters. For all of our simulations, unless specified otherwise, the initial distribution was a 3-dimensional multivariate normal with mean $(0.5, 0.5, 0.5)$, a standard deviation of 0.1 in all variables and zero covariance. First situation where the population sizes were no longer equal was considered. Suppose that

$$\nu_1 = 0.9,\ \nu_2 = 1/3,\ \nu_3 = 1/3,$$

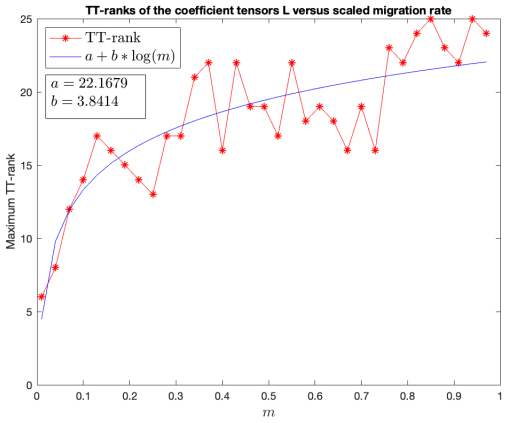
and migration stays symmetric with

$$M_{1\leftarrow 2} = M_{1\leftarrow 3} = 0.01,$$

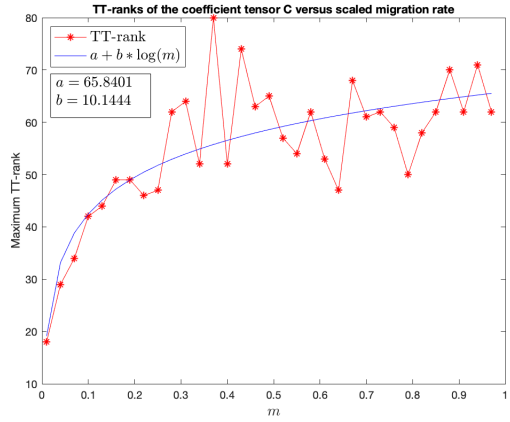
Figure 6.10: Effect of increasing migration rates on TT-ranks.



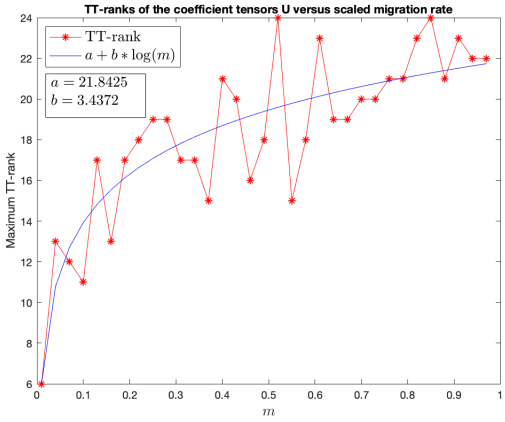
(a) Setup time versus migration rates.



(b) Maximum TT-ranks of $\boldsymbol{\mathcal{C}}$.



(c) Maximum TT-ranks of the $\widehat{\boldsymbol{\mathcal{L}}}_p$'s.



(d) Maximum TT-ranks of the $\widehat{\boldsymbol{\mathcal{U}}}_p$'s.

$$M_{2\leftarrow 1} = M_{2\leftarrow 3} = 0.01,$$
$$M_{3\leftarrow 1} = M_{3\leftarrow 2} = 0.01.$$

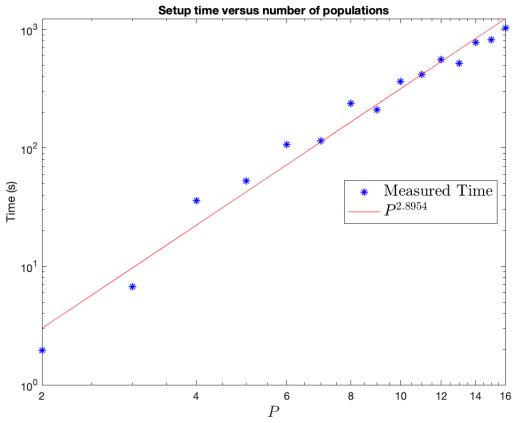
Then the density ϕ and the AFS at $\tau = 0.02$ were estimated, with

$$N_1 = 90 \text{ and } N_2 = N_3 = 30.$$

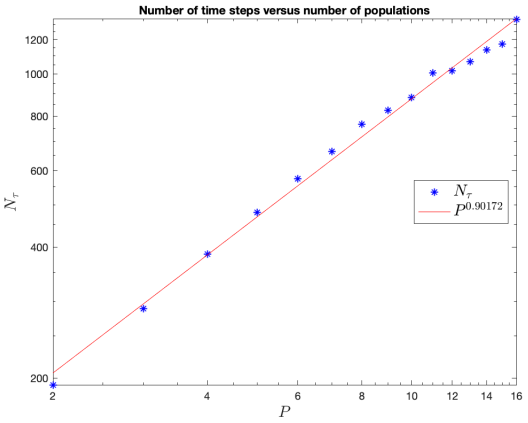
We ran the TT-FVM with $I_\Omega = 80$ control volumes along each population axis using rounded coefficient tensors. To visualise the time evolution of the approximations he marginal densities and the marginal allele frequency spectra are plotted at regular intervals. Marginal densities were calculated by integrating over all populations except one:

$$\phi_1(t, x_1) := \int_0^1 \int_0^1 \phi(t, \boldsymbol{x}) \, \mathrm{d}x_2 \, \mathrm{d}x_3,$$
$$\phi_2(t, x_2) := \int_0^1 \int_0^1 \phi(t, \boldsymbol{x}) \, \mathrm{d}x_1 \, \mathrm{d}x_3,$$
$$\phi_3(t, x_3) := \int_0^1 \int_0^1 \phi(t, \boldsymbol{x}) \, \mathrm{d}x_1 \, \mathrm{d}x_2.$$

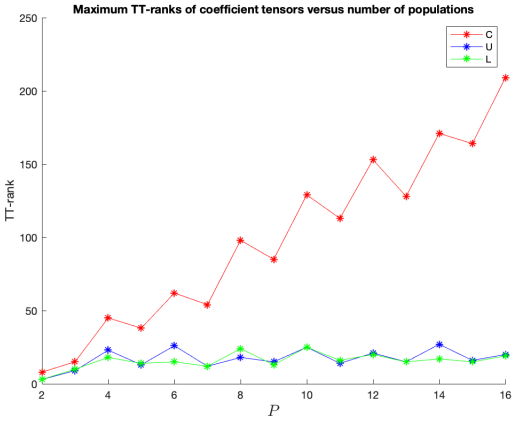
Figure 6.11: Effects of increasing the number of populations.



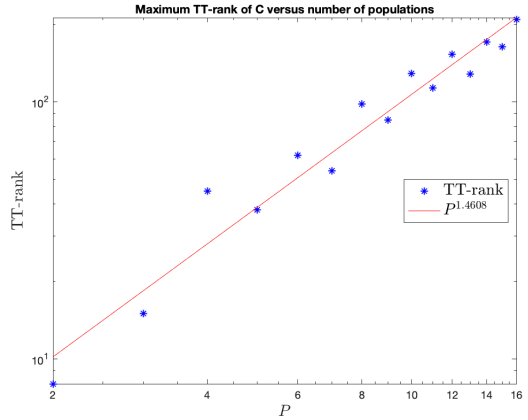
(a) Setup Time.



(b) Number of time steps.



(c) Maximum TT-ranks of the coefficient tensors.



(d) Maximum TT-rank of $\boldsymbol{\mathcal{C}}$.

These integrals are computed using a midpoint rule. Similarly, the marginal AFS is calculated by summing over all populations except one:

$$\boldsymbol{\mathcal{M}}_1(t, d_1) := \sum_{d_2=0}^{N_2} \sum_{d_3=0}^{N_3} \boldsymbol{\mathcal{M}}(t, d_1, d_2, d_3), \ \forall d_1 = 0 \ldots, N_1,$$

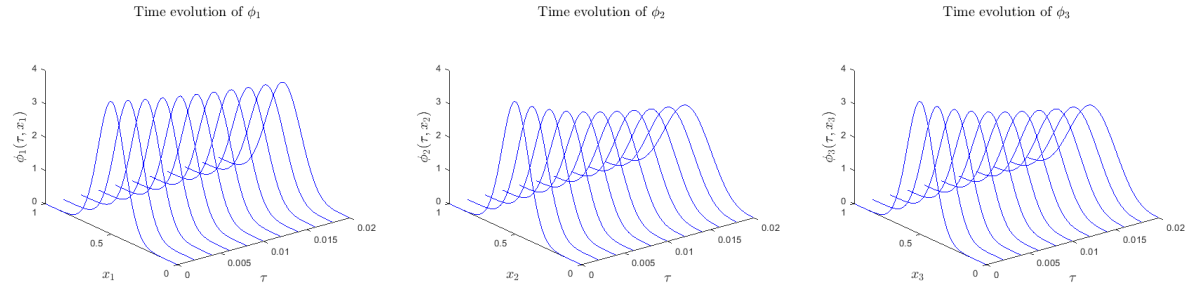
$$\boldsymbol{\mathcal{M}}_2(t, d_2) := \sum_{d_1=0}^{N_1} \sum_{d_3=0}^{N_3} \boldsymbol{\mathcal{M}}(t, d_1, d_2, d_3), \ \forall d_2 = 0 \ldots, N_2,$$

$$\boldsymbol{\mathcal{M}}_3(t, d_3) := \sum_{d_1=0}^{N_1} \sum_{d_2=0}^{N_2} \boldsymbol{\mathcal{M}}(t, d_1, d_2, d_3), \ \forall d_3 = 0 \ldots, N_3.$$

Figure 6.12 and Figure 6.13 show the time evolution of the marginal densities and spectra with the current model parameters. From this simulation we see that that the general shape of the density ϕ is retained in the allele frequency spectrum with scaled down values. Furthermore, we see that the diffusion occurred slower in population 1 than in populations 2 and 3 which have smaller population sizes. This reflects the fact

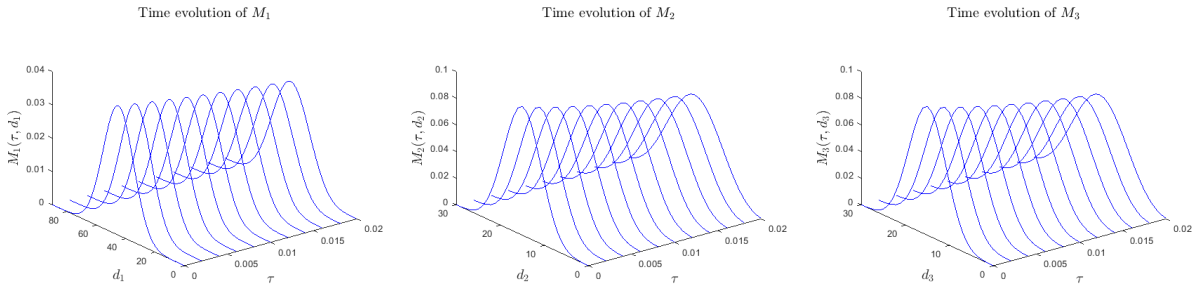
that small changes in allele frequencies will have a smaller effect on the overall frequency distribution in larger populations.

Figure 6.12: Time evolution of marginal densities with one large population



(a) (b) (c)

Figure 6.13: Time evolution of marginal allele frequency spectra with one large population



(a) (b) (c)

To see what would happen if all the population sizes were different, the migration rates from the previous simulation were kept but the population sizes were changed to

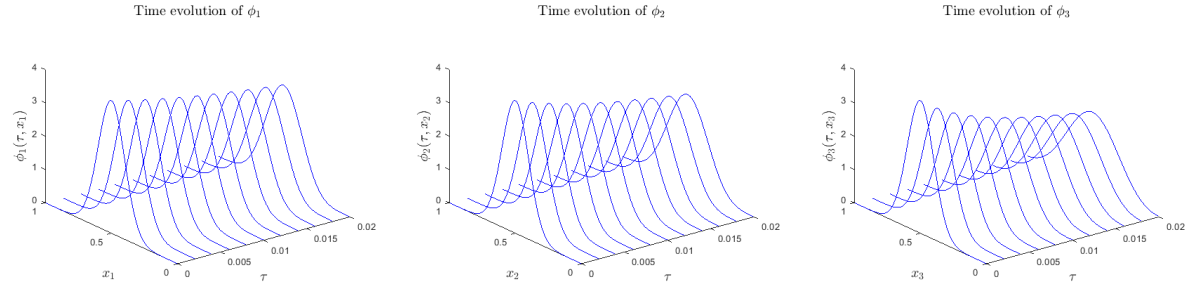
$$\nu_1 = 0.75,\ \nu_2 = 0.5,\ \text{and}\ \nu_3 = 0.25.$$

Population sizes used were

$$N_1 = 100,\ N_2 = 75,\ N_3 = 50.$$

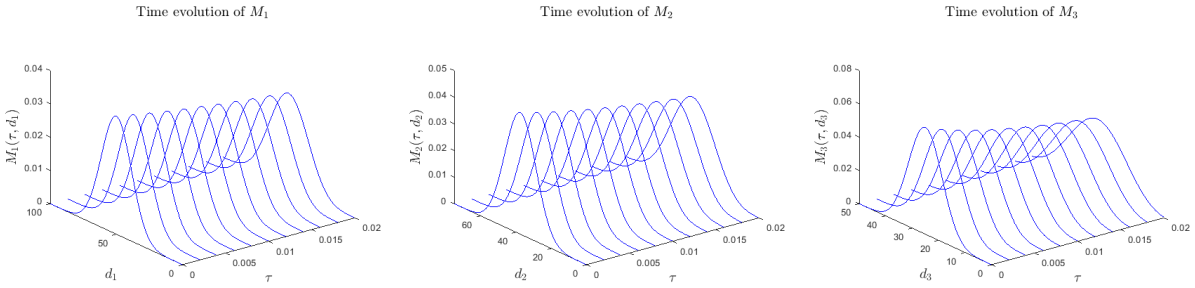
Figure 6.14 and Figure 6.15 show the time evolution of the marginal densities and spectra with the new model parameters. Again, it is seen that the general shape of the density is preserved in the allele frequency spectrum and the larger the population size the slower the diffusion.

Figure 6.14: Time evolution of marginal densities with different population sizes



(a) (b) (c)

Figure 6.15: Time evolution of marginal allele frequency spectra with different population sizes,



(a) (b) (c)

Next the effects of non-symmetric migration rates were considered. The population sizes were chosen to be equal:

$$\nu_1 = \nu_2 = \nu_3 = 1/3.$$

We did not make the population sizes large to remove the effects of diffusion, because the effects of migration are very subtle and are easier to see when there is some diffusion occurring. The migration rates we chose were

$$\begin{aligned} M_{1\leftarrow 2} = 3 \quad M_{1\leftarrow 3} = 3, \\ M_{2\leftarrow 1} = 0.01, \quad M_{2\leftarrow 3} = 0.01, \\ M_{3\leftarrow 1} = 0.01, \quad M_{3\leftarrow 2} = 0.01. \end{aligned}$$

so migration out of the first population was much higher than the migration out of the other two populations. The populations sizes used were

$$N_1 = N_2 = N_3 = 100.$$

Such large migration rates were chosen to exaggerate the effects of migration, a scaled migration rate of 3 is also not very large in a species with a large effective population. A larger time $\tau = 0.03$ was also used to give the effects of migration a better chance of being visible. Figure 6.16 and Figure 6.17 show the time evolution of the marginal densities and spectra with the new model parameters. In this case the effects of increased migration rates can't be so easily seen by looking at the time evolution of the marginal densities and spectra as we have before. So, we also kept track of the

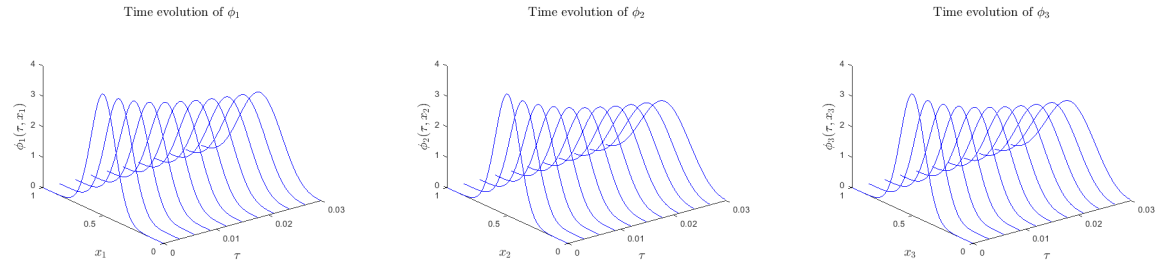
maximum values of the marginal distributions and spectra, these are shown in Figure-6.18 and Figure-6.19. In these figures we can see that the maximum of population 1 decays slightly slower than the maxima in populations 2 and 3. This happens because all three populations have the same initial distribution, and the peak does not move in the time period over which we run the simulation. When we view the time evolution of the AFS as a diffusion process, new chromosomes enter population 1 from population 2 and 3 via a flow of density from populations 2 and 3 to population 1. Since the densities of all populations are concentrated around the same point, most of the density will enter population 1 at the peak of its distribution. This slows down the decay of the peak as the distribution spreads out along the x_1 axis. This suggests that if population 2 and 3 have initial distributions that are peaked in a different position than population 1, we should see the peak of ϕ_1 shift over time.

Finally, a different initial distribution was chosen. The initial distribution was chosen to be a multivariate normal with mean $(0.35, 0.65, 0.65)$ and standard deviation of 0.05 in each variable with zero covariance. Population sizes were also increased to

$$\nu_1 = \nu_2 = \nu_3 = 0.9$$

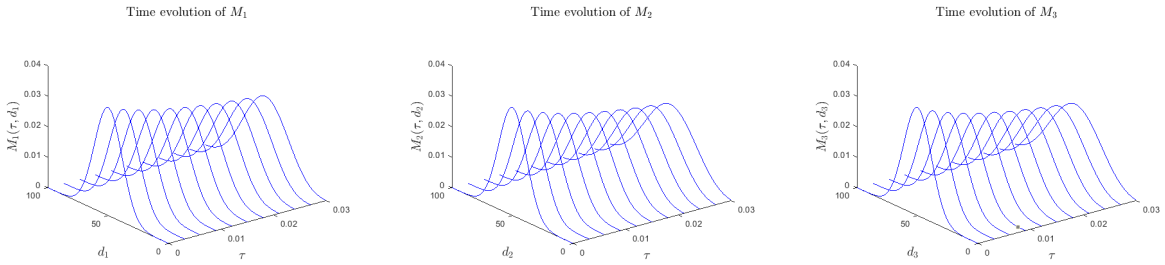
and the time to $\tau = 0.04$. The increased population size reduces the diffusion which makes the peaks more clear and by increasing τ the peak is allowed more time to shift. In Figure 6.20 and Figure 6.21 the initial and final distributions are shown. Exactly the behaviour that was predicted is seen in these plots. Over time ϕ_1 moves to the right as new chromosomes enter population 1 from populations 2 and 3.

Figure 6.16: Time evolution of marginal densities with non-symmetric migration rates



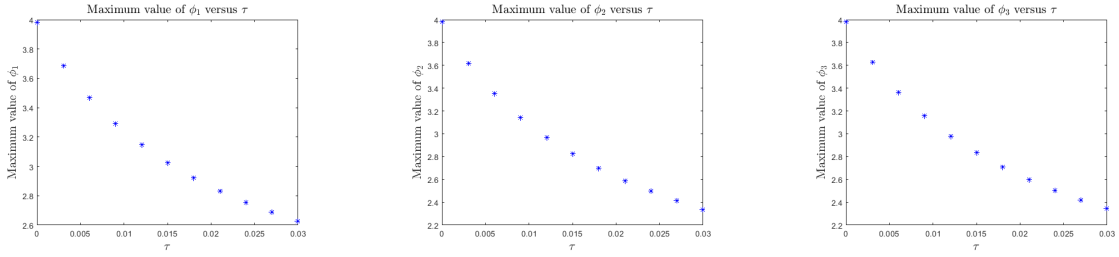
(a) (b) (c)

Figure 6.17: Time evolution of marginal allele frequency spectra with non-symmetric migration rates



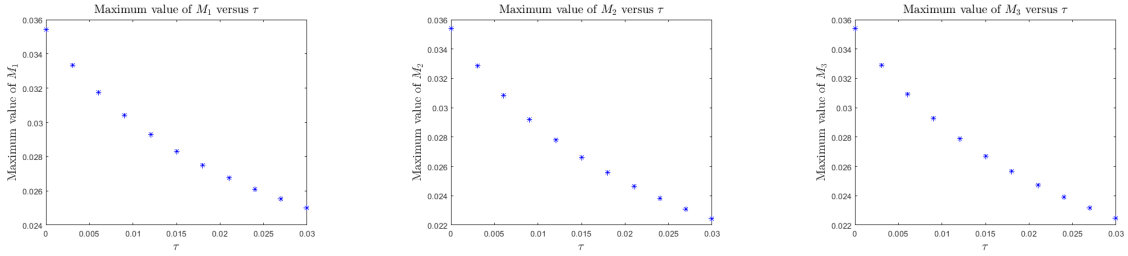
(a) (b) (c)

Figure 6.18: Maximum values of the marginal densities



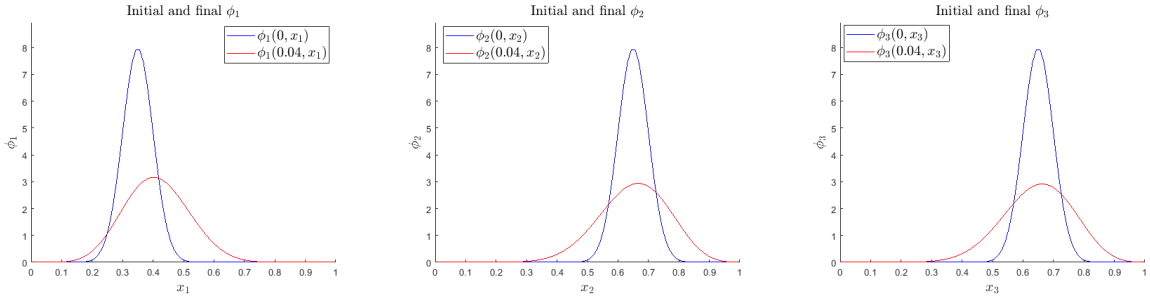
(a) (b) (c)

Figure 6.19: Maximum values of the marginal allele frequency spectra.



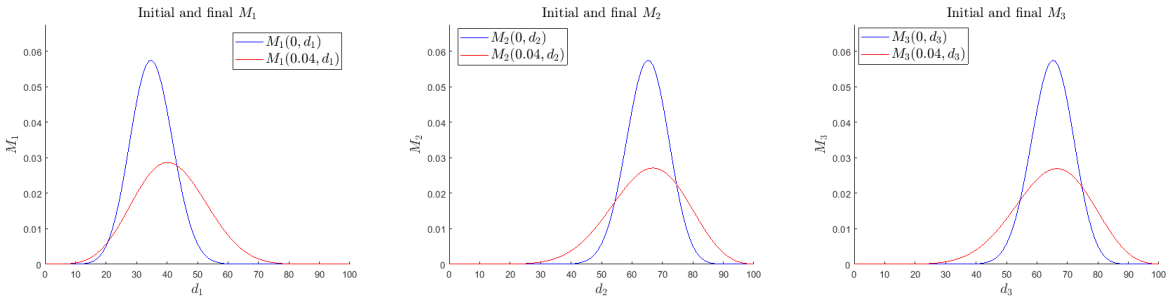
(a) (b) (c)

Figure 6.20: Initial and final marginal densities with an asymmetric initial distribution



(a) (b) (c)

Figure 6.21: Initial and final marginal spectra with an asymmetric initial distribution.



(a) (b) (c)

6.4 Comparison to Existing Methods

We have now provided experimental evidence that our method of approximating the AFS in three populations works. Some understanding of how the computation time is affected by the model parameters has also been gained. To finish, we compare the results of our analysis to those of other numerical methods which have been applied to the same problem.

There are two standard methods for numerically approximating the allele frequency spectrum by solving the Wright-Fisher diffusion equation. The most widely used method is implemented in a Python software package known as ∂a∂i, based on the 2009 paper [18] by Gutenkunst et al. Their approach implements a finite difference scheme to discretise the convection-diffusion equation and is optimised for 2 to 3 populations but can simulate up to 6 populations. In 3 populations ∂a∂i is able to approximate the AFS using the same parameters that we used in Section 6.3.2 using 320 data points along each population axis in 44.6727 seconds, but our method took about 15 minutes when using rounded coefficient tensors. Using 640 data points alone each population axis ∂a∂i took 421.2287 seconds, while our method took about 1 hour and 50 minutes.

The other standard method is based on truncated expansions of the solution ϕ in complete bases of a certain functional space. The theory for this method was developed by Lukic et al. in the 2011 paper [34] and in their 2012 paper [33] Lukic et al. was able to successfully simulate up to 6 populations. Lukic et al's method was able to approximate the AFS using 35 basis polynomials in 57.41 seconds. Lukic et al. also compared their spectral method with ∂a∂i, which was able to simulate the AFS in 6 populations in about 250 seconds using 300 grid points along each population axis. When simulating 6 populations with 300 control volumes along each population axis our method would take multiple hours.

While our method does not appear to out-perform current methods in this context, to our knowledge it was the first attempt at using the TT-format to approximate solutions to the Wright-Fisher diffusion equation, an advance which opens the possibility of linear complexity increases with respect to dimension. As we saw in Section 6.3.1, once we have the density ϕ in TT-format we can compute the AFS extremely quickly and efficiently. In Section 6.3.3 we also saw that the main driver of increased computation times was the effect that model parameters had on the number of time steps required. This has nothing to do with the TT-format but is entirely due to the finite volume method, so other implementations using our finite volume method will also be held back. The TT-format implementation of our finite volume method was also held back by the presence of maximum and minimum functions in the coefficient tensors. These maximum and minimum value functions require more computation time to evaluate, and they increase the ranks of the coefficient tensors. So overall, our method's success was held back by our use of the finite volume method. This makes us hopeful that future attempts using different discretisation schemes of convection-diffusion equations might be more successful.

Chapter 7

Conclusions and Future Work

We conclude this thesis with a summary of the key contributions and results discussed in the main body, and we mention possible future improvements that can be made.

7.1 Conclusions

The main aim of this thesis was to explore the TT-format techniques for implementing a finite volume method to solve general time-homogeneous convection diffusion equations with zero source terms. We then applied our method to an application in population genetics.

In Chapters 2 and 3 we presented overviews of the two central mathematical tools used in this thesis. The general idea of finite volume methods was discussed in Chapter 2 while 3 discussed tensors and tensor decompositions.

In Chapter 4 we derived a finite volume method for general time-homogeneous convection-diffusion equations with zero source terms using rectangular control volumes. It was proven that this method preserves the positivity and integral of the initial value function. The method was tested by implementing it for a two-dimensional problem, in this example evidence that our method converges as we refine the grid of control volumes was found.

In Chapter 5 we developed a TT format algorithm for implementing our finite volume method which we called the TT-FVM algorithm. The TT-FVM algorithm took model parameters (domain, final time and coefficient functions) and numerical parameters (number of control volumes along each dimension) and returned a discrete approximation of the solution in TT format. Our TT-FVM algorithm scaled as $\mathcal{O}(D^4)$ with D being the number of spatial dimensions of the convection-diffusion equation. To implement the TT-FVM algorithm it was necessary to develop a circular index shifting algorithm for the TT format. We named this algorithm the TT-circshift algorithm. The TT-circshift algorithm took a tensor in TT format as an input and returned the tensor with shifted indices in TT format without having to perform any additional TT-SVDs or cross approximations. To our knowledge such an algorithm has not been published before, and there is no equivalent algorithm currently available in the TT-toolbox package.

In Chapter 6 the TT-FVM algorithm was used to approximate the solution to the Wright-Fisher diffusion equation for the Allele Frequency Spectra in three populations. We were able to provide evidence that the approximations generated by the TT-FVM method converge, although there remains a possibility that the methods are not converging to a solution of the Wright-Fisher diffusion equation. A new technique was developed for calculating the expected allele frequency spectrum in TT format if the solution to the Wright-Fisher diffusion equation is approximated in TT format. To do this we developed a method for creating a TT format approximation of the so-called Binomial Tensor. The new method is much more efficient and accurate than a standard cross approximation. Furthermore, this new method for approximating the Binomial tensor should be usable for an arbitrary number of populations, but it cannot handle population sizes larger than 170. We also chose to round the coefficient tensors in the TT-FVM algorithm, which significantly improved computation time without significantly changing the approximation formed by the algorithm.

In Chapter 6 an in depth analysis of the impact that model parameters have on the TT-FVM algorithm was also performed. We found that small relative population sizes can significantly increase the computation time of our method. In contrast, we found that migration rates do not impact computation time as significantly as relative population sizes. We were also able to look into the qualitative effects that model parameters have on the approximations formed by the TT-FVM algorithm. It was found that large relative population sizes correspond to slow diffusion and small relative population sizes speed up diffusion.
Migration rates control how much distributions across different populations mix over time.

Finally, we briefly compared the performance of the TT-FVM for solving the Wright-Fisher diffusion equation to two other prominent methods. Both of these methods appeared to outperform our TT-FVM algorithm.

7.2 Future Work

7.2.1 Prove that our Finite Volume Method is Convergent

Proving that the finite volume method developed in this thesis is convergent is an important step in establishing it as a viable tool for numerically solving convection-diffusion equations that appear in other areas of research. Only experimental evidence that our finite volume method converges in a few examples has been provided. It has not been proven analytically that our method is always convergent. A subset of equations for which it is possible to show that our method is convergent has not been found. If there exists appropriate existence and uniqueness theorems for a subset of equations, we suspect that by following the proof of convergence given in [38] it may be possible to prove that our finite volume method is convergent for this subset of equations.

In [9, Sec. 17.1] a finite volume method similar to the one developed in Chapter 4 shown to be convergent on bounded domains if Dirichlet boundary values are imposed. It may be possible to adapt the theory provided there to unbounded domains or to different

boundary conditions.

7.2.2 Reformulating the Integration Step

The first place where we think an improvement can be made to our method is in the integration step of the TT-FVM. We time step towards the solution at the final time by the recurrence

$$\overline{\boldsymbol{\Phi}}^{n+1} = \widehat{\boldsymbol{\mathcal{M}}} * \overline{\boldsymbol{\Phi}}^{n} + \sum_{d=1}^{D} \left(\tau_{d,-1}^{circ}[\widehat{\boldsymbol{\mathcal{U}}}_d * \overline{\boldsymbol{\Phi}}^{n}] + \tau_{d,1}^{circ}[\widehat{\boldsymbol{\mathcal{L}}}_d * \overline{\boldsymbol{\Phi}}^{n}] \right), \tag{7.1}$$

as shown in Proposition 5.0.3. Computing one step in this recurrence requires multiple operations, each of which increases the TT ranks of the resultant tensors. This requires us to round the approximation $\overline{\boldsymbol{\Phi}}^{n+1}$ at the end of each time step. All of this increases the computational cost of our method. If we can reformulate (7.1) in a way that requires less operations and reduces the rank increase it might be possible to significantly improve the speed of our method.

7.2.3 A Different Discretisation Scheme

In Section 6.3.3 we posited that our TT-FVM algorithm as tool for solving the Wright-Fisher diffusion equation was not held back by the use of the TT-format, but rather by our use of the finite volume method. The TT-ranks of the coefficient tensors were increased by the minimum and maximum functions appearing in their definition. The number of time steps required to preserve positivity also grew too rapidly with dimension and decreasing population sizes.

Stable and convergent finite difference schemes for convection-diffusion equations already exist [1] (in this paper they consider a more general equation known as the *Fokker-Planck* equation). Moreover, finite difference approximations of first and second order derivatives can be easily implemented in the TT-format with small ranks as outlined in Section 3.2 of [44]. This makes us hopeful that it is possible to greatly improve on the TT-FVM algorithm by switching to a finite difference based discretisation rather than a finite volume based discretisation. It is, however, still unclear that making the switch to finite difference based methods will solve the problem we had with the growing number of time steps required to preserve positivity. However, we do not have to preserve positivity by controlling the temporal step-size. By adjusting the convection-diffusion equation so that it is solved by the square root of the solution to the original partial differential equation and then squaring the approximation to the new equation might also be a viable method of preserving positivity.

7.2.4 Approximating the Binomial Tensor for Larger Population Sizes

The benefit of our method for approximating the binomial tensor is that it was done using midpoint approximations of the binomial distribution function. This means that it is not restricted to being used with finite volume methods. The grid of points at which we approximate the binomial distribution function can easily be shifted to coincide with

a grid of points used in a finite difference based discretisation of the Wright-Fisher diffusion equation. The main limitation of the method is not the number of populations, but rather the population sizes since it fails for populations larger than 170. The method fails because the binomial distribution function becomes too small for populations larger than 170 and is indistinguishable from the zero function to a machine running a maxvol algorithm. One way in which we think that this issue can be circumvented is to approximate a scaled-up version of the binomial distribution function by composing it with another function.

7.2.5 Finding Analytic Solutions to the Wright-Fisher Diffusion Equation

To date, no exact analytic solutions have been found for the Wright-Fisher diffusion equation for more than one population. The solution found by Kimura in [25] is also not a solution on a bounded domain, but a solution on all of $\mathbb{R}$. There are also no complete treatments of the boundary conditions of the Wright-Fisher diffusion equation when we restrict the domain to $(0, 1)^P$. These two topics should be explored in the future.

Bibliography

[1] J. S. Chang and G. Cooper, *A practical difference scheme for fokker-planck equations*, Journal of Computational Physics **6** (1970), no. 1, 1–16.

[2] B. Charlesworth, *Effective population size and patterns of molecular evolution and variation*, Nature Reviews Genetics **10** (2009), no. 3, 195–205.

[3] P. C. Chatwin and C. M. Allen, *Mathematical models of dispersion in rivers and estuaries*, Annual Review of Fluid Mechanics **17** (1985), no. 1, 119–149.

[4] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM journal on Matrix Analysis and Applications **21** (2000), no. 4, 1253–1278.

[5] V. De Silva and L. H. Lim, *Tensor rank and the ill-posedness of the best low-rank approximation problem*, SIAM Journal on Matrix Analysis and Applications **30** (2008), no. 3, 1084–1127.

[6] S. Dolgov, K. Anaya-Izquierdo, C. Fox, and R. Scheichl, *Approximation and sampling of multivariate probability distributions in the tensor train decomposition*, Statistics and Computing **30** (2020), no. 3, 603–625.

[7] R. Durrett, *Probability models for DNA sequence evolution*, Springer Science & Business Media, 2008.

[8] C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika **1** (1936), no. 3, 211–218.

[9] R. Eymard, T. Gallouët, and R. Herbin, *Finite volume methods*, Handbook of numerical analysis **7** (2000), 713–1018.

[10] J.H. Ferziger, M. Peric, and R.L Street, *Computational Methods for Fluid Dynamics*, Fourth edition., Springer, Cham, Switzerland, 2020 (eng).

[11] R. A. Fisher, *XXI.-On The Dominance Ratio*, Proceedings of the Royal Society of Edinburgh **42** (1923), 321–341.

[12] H. C. Goicoechea, S. Yu, A. C. Olivieri, and A. D. Campiglia, *Four-Way Data Coupled to Parallel Factor Model Applied to Environmental Analysis: Determination of 2, 3, 7, 8-Tetrachloro-dibenzo-para-dioxin in Highly Contaminated Waters by Solid- Liquid Extraction Laser-Excited Time-Resolved Shpol'skii Spectroscopy*, Analytical chemistry **77** (2005), no. 8, 2608–2616.

[13] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, *A theory of pseudoskeleton approximations*, Linear algebra and its applications **261** (1997), no. 1-3, 1–21.

[14] S.A. Goreinov, I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, *How to find a good submatrix*, Matrix Methods: Theory, Algorithms And Applications: Dedicated to the Memory of Gene Golub, 2010, pp. 247–256.

[15] L. Grasedyck, *Hierarchical singular value decomposition of tensors*, SIAM Journal on Matrix Analysis and Applications **31** (2010), no. 4, 2029–2054.

[16] L. Grasedyck and W. Hackbusch, *An introduction to hierarchical (H-) rank and TT-rank of tensors with examples*, Computational Methods in Applied Mathematics Comput. Methods Appl. Math. **11** (2011), no. 3, 291–304.

[17] B. Gustafsson, *Order of accuracy and the convergence rate*, High Order Difference Methods for Time Dependent PDE (2008), 69–80.

[18] R. N. Gutenkunst, R. D. Hernandez, S. H. Williamson, and C. D. Bustamante, *Inferring the joint demographic history of multiple populations from multidimensional SNP frequency data*, PLoS genetics **5** (2009), no. 10, e1000695.

[19] R. Harshman, *Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-model factor analysis*, 1970.

[20] D.L. Hartl and A.G. Clark, *Principles of Population Genetics*, 4th ed., Sinauer Associates, Sunderland, Mass., 2007 (eng).

[21] J. Håstad, *Tensor rank is NP-Complete*, International Colloquium on Automata, Languages, and Programming, 1989, pp. 451–460.

[22] P.W. Hedrick, *Genetics of populations*, 4th ed., Jones and Bartlett Publishers, Sudbury, Mass., 2011 (eng).

[23] J. Jouganous, W. Long, A. P. Ragsdale, and S. Gravel, *Inferring the Joint Demographic History of Multiple Populations: Beyond the Diffusion Approximation*, Genetics **206** (2017), no. 3, 1549–1567.

[24] M. Kimura, *Stochastic processes and distribution of gene frequencies under natural selection*, Cold Spring Harb Symp Quant Biol **20** (1955), 33–53.

[25] ———, *Diffusion models in population genetics*, Journal of Applied Probability **1** (1964), no. 2, 177–232.

[26] N. Kishore Kumar and J. Schneider, *Literature survey on low rank approximation of matrices*, Linear and Multilinear Algebra **65** (2017), no. 11, 2212–2244.

[27] T. G. Kolda, *Orthogonal tensor decompositions*, SIAM Journal on Matrix Analysis and Applications **23** (2001), no. 1, 243–255.

[28] ———, *Multilinear Operators for Higher-Order Decompositions.*, Technical Report SAND2006-2081, Sandia National Laboratories, 2006.

[29] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM review **51** (2009), no. 3, 455–500.

[30] O. Kolditz, *Computational Methods in Environmental Fluid Mechanics*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002 (eng).

[31] K. Lange, *Applied probability*, Springer texts in statistics, Springer, New York, 2003 (eng).

[32] G. Lukaszewicz and P. Kalita, *Navier-Stokes Equations: An Introduction with Applications*, Springer, Cham, 2016.

[33] S. Lukić and J. Hey, *Demographic inference using spectral methods on SNP data, with an analysis of the human out-of-Africa expansion*, Genetics **192** (2012), no. 2, 619–639.

[34] S. Lukić, J. Hey, and K. Chen, *Non-equilibrium allele frequency spectra via spectral methods*, Theoretical population biology **79** (2011), no. 4, 203–219.

[35] P. A. Markowich, C. A. Ringhofer, and C. Schmeiser, *The drift diffusion equations*, Springer Vienna, Vienna, 1990.

[36] MATLAB, *version 9.6 (R2019a)*, The MathWorks Inc., Natick, Massachusetts, 2019.

[37] S. Matsumura and P. Forster, *Generation time and effective population size in Polar Eskimos*, Proceedings of the Royal Society B: Biological Sciences **275** (2008), no. 1642, 1501–1508.

[38] B. Merlet and J. Vovelle, *Error estimate for finite volume scheme*, Numerische Mathematik **106** (2006), no. 1, 129–155.

[39] A. Muñoz de la Peña, A. Espinosa Mansilla, D. González Gómez, A. C. Olivieri, and H. C. Goicoechea, *Interference-free analysis using three-way fluorescence data and the parallel factor model. Determination of fluoroquinolone antibiotics in human serum*, Analytical chemistry **75** (2003), no. 11, 2640–2646.

[40] R. A. Norton, C. Fox, and M. E. Morrison, *Numerical Approximation of the Frobenius–Perron Operator using the Finite Volume Method*, SIAM Journal on Numerical Analysis **56** (2018), no. 1, 570–589.

[41] A. C. Olivieri and K. Faber, *New developments for the sensitivity estimation in four-way calibration with the quadrilinear parallel factor model*, Analytical chemistry **84** (2012), no. 1, 186–193.

[42] I. Oseledets, *Personal communication*, 2020.

[43] I. V. Oseledets and E.E. Tyrtyshnikov, *TT-cross approximation for multidimensional arrays*, Linear Algebra and its Applications **432** (2010), no. 1, 70–88.

[44] I.V. Oseledets, *Tensor-Train Decomposition*, SIAM Journal on Scientific Computing **33** (2011), no. 5, 2295–2317.

[45] ———, *tt-toolbox*, Mathworks, 2020. [Online; accessed 2020].

[46] I.V. Oseledets and E. E. Tyrtyshnikov, *Breaking the curse of dimensionality, or how to use SVD in many dimensions*, SIAM Journal on Scientific Computing **31** (2009), no. 5, 3744–3759.

[47] P. Paatero, *Construction and analysis of degenerate PARAFAC models*, Journal of Chemometrics: A Journal of the Chemometrics Society **14** (2000), no. 3, 285–299.

[48] D. Poole, *Linear algebra: A modern introduction*, Cengage Learning, 2014.

[49] B. E. Rapp, *Microfluidics : Modeling Mechanics and Mathematics*, First edition., Micro and Nano technologies, William Andrew, Amsterdam, [Netherlands], 2017 (eng).

[50] D. Saari, *Mathematics of finance : an intuitive introduction*, Undergraduate texts in mathematics, Springer, Cham, 2019 (eng).

[51] D. V. Savostyanov, *Quasioptimality of maximum-volume cross interpolation of tensors*, Linear Algebra and its Applications **458** (2014), 217–244.

[52] R. L. Schilling, *Measures, integrals and martingales*, Second edition., Cambridge University Press, Cambridge, United Kingdom ; New York, NY, 2017 (eng).

[53] S. T. Sonis, *Genomics, Personalized Medicine and Oral Disease*, Springer, 2015.

[54] J. Stewart, *Calculus*, 7th ed., International Metric Version., Brooks/Cole Cengage Learning, Pacific Grove, CA, 2012 (eng).

[55] D. W. Stroock, *Multidimensional Diffusion Processes*, Reprint of the 1997 ed., Classics in Mathematics, Springer Berlin Heidelberg : Imprint: Springer, Berlin, Heidelberg, 2006 (eng).

[56] T. D. Tran, J. Hofrichter, and J. Jost, *An introduction to the mathematical structure of the Wright–Fisher model of population genetics*, Theory in Biosciences **132** (2013), no. 2, 73–82.

[57] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, Vol. 50, SIAM, 1997.

[58] L. R. Tucker, *The extension of factor analysis to three-dimensional matrices*, Contributions to mathematical psychology., 1964, pp. 110–127.

[59] ———, *Some mathematical notes on three-mode factor analysis*, Psychometrika **31** (1966), no. 3, 279–311.

[60] L.R. Tucker, *Implications of factor analysis of three-way matrices for measurement of change*, Problems in measuring change **15** (1963), 122–137.

[61] E. Tyrtyshnikov, *Incomplete cross approximation in the mosaic-skeleton method*, Computing **64** (2000), no. 4, 367–380.

[62] L.A. Urry, N. Meyers, M.L. Cain, S.A. Wasserman, P.V. Minorsky, and J.B. Reece, *Campbell biology*, 11th edition. Australian and New Zealand version., Pearson Australia Melbourne, VIC, 2017 (English).

[63] C. F. Van Loan, *The ubiquitous Kronecker product*, Journal of computational and applied mathematics **123** (2000), no. 1-2, 85–100.

[64] G. Van Rossum and F. L. Drake, *Python 3 reference manual*, CreateSpace, Scotts Valley, CA, 2009.

[65] S. Wright, *Evolution in mendelian populations*, Genetics **16** (1931), no. 2, 97–159.

[66] ———, *The differential equation of the distribution of gene frequencies*, Proceedings of the National Academy of Sciences of the United States of America **31** (1945), no. 12, 382.

[67] B. Zohuri, *Thermal-Hydraulic Analysis of Nuclear Reactors*, 2nd ed., Springer, Cham, 2017 (eng).

Appendix A

Auxiliary results: Tensors

A.0.1 n-Mode Products in the TT-format

Proposition A.0.1 (Proposition 3.2.11.3)
Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ be given in TT-format with cores $\boldsymbol{\mathcal{X}}_1, \ldots, \boldsymbol{\mathcal{X}}_N$ and TT-ranks $R_1, \ldots, R_N$. Let $\boldsymbol{M} \in R^{J \times R_n}$ for some $n = 1, \ldots, N$. Then the n-mode product $\boldsymbol{\mathcal{Z}} := \boldsymbol{\mathcal{X}} \times_n \boldsymbol{M}$ has a TT-decomposition with ranks $R_1^X, \ldots, R_N^X$ and cores given by

$$\boldsymbol{\mathcal{Z}}_l = \boldsymbol{\mathcal{X}}_l, \quad \textit{for all} \;\; l \neq n,$$

and

$$\boldsymbol{\mathcal{Z}}_n(r_{n-1}, :, :) = \boldsymbol{M}\boldsymbol{\mathcal{X}}(r_{n-1}, :, :), \quad \textit{for} \;\; r_{n-1} \in \{1, \ldots, R_{n-1}\}.$$

It then follows that computing the n-mode product requires $\mathcal{O}(JIR^2)$ operations.

Proof:
Recall that the n-mode product of $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{M}$ is the tensor given by

$$(\boldsymbol{\mathcal{X}} \times_n \boldsymbol{M})(i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_N) = \sum_{i_n=1}^{I_n} \boldsymbol{M}(j, i_n)\boldsymbol{\mathcal{X}}(i_1, \ldots, i_{n-1}, i_n, i_{n+1}, \ldots, i_N). \tag{A.1}$$

Substituting the TT-format of $\boldsymbol{\mathcal{X}}$ as given by (3.6) into (A.1) we find

$$\begin{aligned}
&(\boldsymbol{\mathcal{X}} \times_n \boldsymbol{M})(i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_N) \\
&\quad = \sum_{i_n=1}^{I_n} \boldsymbol{M}(j, i_n)\boldsymbol{\mathcal{X}}_1(:, i_1, :) \cdots \boldsymbol{\mathcal{X}}_{n-1}(:, i_{n-1}, :)\boldsymbol{\mathcal{X}}_n(:, i_n, :)\boldsymbol{\mathcal{X}}_{n+1}(:, i_{n+1}, :) \cdots \boldsymbol{\mathcal{X}}_N(:, i_N, :) \\
&\quad = \boldsymbol{\mathcal{X}}_1(:, i_1, :) \cdots \boldsymbol{\mathcal{X}}_{n-1}(:, i_{n-1}, :) \left[\sum_{i_n=1}^{I_n} \boldsymbol{M}(j, i_n)\boldsymbol{\mathcal{X}}_n(:, i_n, :) \right] \boldsymbol{\mathcal{X}}_{n+1}(:, i_{n+1}, :) \cdots \boldsymbol{\mathcal{X}}_N(:, i_N, :).
\end{aligned}$$

From this we see that $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{M}$ is an Nth-order tensor with the same mode lengths as $\boldsymbol{\mathcal{X}}$ except the n-th mode now has J entries rather than I_n. Furthermore, $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{M}$ is also already in TT-format with all cores except the nth one remaining unchanged. The TT-cores of $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{M}$ are given by

$$\widehat{\boldsymbol{\mathcal{X}}}_n = \begin{cases} \boldsymbol{\mathcal{X}}_l & \text{if } l \neq n, \\ \sum_{i_n=1}^{I_n} \boldsymbol{M}(:, i_n)\boldsymbol{\mathcal{X}}_n(:, i_n, :) & \text{if } l = n. \end{cases}$$

We can make this calculation even simpler by considering the sum over i_n in the nth core more closely. The third-order tensor $\widehat{\boldsymbol{\mathcal{X}}}_n$ has dimensions $R_{n-1} \times J \times R_n$, and for fixed $j \in \{1, \ldots, J_n\}$, $r_{n-1} \in \{1, \ldots, R_{n-1}\}$ and $r_n \in \{1, \ldots, R_n\}$ we have

$$\begin{aligned}\widehat{\boldsymbol{\mathcal{X}}}_n(j, r_{n-1}, r_n) &= \sum_{i_n=1}^{I_n} \boldsymbol{M}(j, i_n)\boldsymbol{\mathcal{X}}_n(r_{n-1}, i_n, r_n) \\ &= \overline{\boldsymbol{X}}_{r_{n-1}}(j, r_n),\end{aligned}$$

where matrices $\overline{\boldsymbol{X}}_{r_{n-1}} \in \mathbb{R}^{R_{n-1}J \times R_n}$ are defined by

$$\overline{\boldsymbol{X}}_{r_{n-1}} = \boldsymbol{M}\boldsymbol{\mathcal{X}}_n(r_{n-1}, :, :).$$

From which it follows that for any fixed $r_{n-1} \in \{1, \ldots, R_{n-1}\}$ we have

$$\widehat{\boldsymbol{\mathcal{X}}}_n(r_{n-1}, :, :) = \boldsymbol{M}\boldsymbol{\mathcal{X}}_n(r_{n-1}, :, :). \tag{A.2}$$

One of these matrix products require $\mathcal{O}(J_n I_n R_n)$ operations and we have to compute R_{n-1} of the matrix products. In total computing the n-mode product of $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{A}$ will require $\mathcal{O}(J_n I_n R_n R_{n-1})$ operations. In general we can say that computing any n-mode product will require $\mathcal{O}(JIR^2)$ operations.

□

Remark A.0.2
It is possible to show from the index from of the matrix product (A.2) that the nth core of the n-mode product is $\boldsymbol{\mathcal{X}}_n \times_2 \boldsymbol{M}$. However, due to fast matrix multiplication algorithms it is more efficient to leave it in the matrix product form.

A natural algorithm that falls out of the proof of Proposition A.0.1 is presented in Algorithm A1. By inspecting the `ttm.m` function file in the TT-toolbox, we can also see that this is the algorithm used in the TT-toolbox,[45].

Algorithm A1: n-mode product in TT-format

Input: An Nth-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ in TT-format, an integer $n \in \{1, \ldots, N\}$ and a matrix $\boldsymbol{M} \in \mathbb{R}^{J \times I_n}$.

Result: The tensor $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{M}$ in TT-format with TT-ranks equal to the TT-ranks of $\boldsymbol{\mathcal{X}}$

Initialisation:

1 Let $\boldsymbol{\mathcal{X}}_1, \ldots, \boldsymbol{\mathcal{X}}_N$ be the TT-cores of $\boldsymbol{\mathcal{X}}$ and $R_0, \ldots, R_N$ the TT-ranks of $\boldsymbol{\mathcal{X}}$.

Main Algorithm:

2 **for** $r_{n-1} = 1, \ldots, R_{n-1}$ **do**

3 | Define $\boldsymbol{X}r_{n-1} := \boldsymbol{M}\boldsymbol{\mathcal{X}}_n(r_{n-1}, :, :)$.

4 **end**

5 Define $\widehat{\boldsymbol{\mathcal{X}}_n}$ by $\widehat{\boldsymbol{\mathcal{X}}_n}(r_{n-1}, :, :) = \boldsymbol{X}_{r_{n-1}}$.

6 **Return**: $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{M}$ in TT-format with cores $\boldsymbol{\mathcal{X}}_1, \ldots, \boldsymbol{\mathcal{X}}_{n-1}, \widehat{\boldsymbol{\mathcal{X}}}_n, \boldsymbol{\mathcal{X}}_{n+1}, \ldots, \boldsymbol{\mathcal{X}}_N$.

A.0.2 Comparing TT-FVM Approximations with different spatial step-sizes

Suppose that ϕ solves a convection-diffusion equation on some rectangular domain Ω in $\mathbb{R}^D$. Partition Ω by control volumes $\Omega(i_1, \ldots, i_D)$ with spatial step-sizes $\Delta x_1, \ldots, \Delta x_D$ (i_d range from 1 to some positive integer I_d for each $d \in \{1, \ldots, D\}$). Let $\overline{\boldsymbol{\Phi}}_1 \in \mathbb{R}^{I_1 \times \ldots \times I_D}$ be a finite volume approximation of ϕ at time T generated by the TT-FVM algorithm on the control volumes $\Omega(i_1, \ldots, i_D)$. Define the piecewise constant function

$$\begin{aligned} &\phi_1 : \Omega \to \mathbb{R} \\ &\phi_1(\boldsymbol{x}) = \overline{\boldsymbol{\Phi}}_1(i_1, \ldots, i_D),\ \forall \boldsymbol{x} \in \Omega(i_1, \ldots, i_D),\ \forall i_1, \ldots, i_D \end{aligned} \tag{A.3}$$

Also, let $\overline{\boldsymbol{\Phi}}_2 \in \mathbb{R}^{2I_1 \times \ldots \times 2I_D}$ be a finite volume approximation of ϕ at time T generated by the TT-FVM algorithm on the control volumes $\widehat{\Omega}(j_1, \ldots, j_D)$ with spatial step-sizes $\Delta x_1/2, \ldots, \Delta x_D/2$ (the index j_d range from 1 to some positive integer $2I_d$ for each $d \in \{1, \ldots, D\}$). Now define the piecewise constant function

$$\begin{aligned} &\phi_2 : \Omega \to \mathbb{R} \\ &\phi_2(\boldsymbol{x}) = \overline{\boldsymbol{\Phi}}_2(j_1, \ldots, j_D),\ \forall \boldsymbol{x} \in \Omega(j_1, \ldots, j_D),\ \forall j_1, \ldots, j_D \end{aligned} \tag{A.4}$$

The L^2 global error between the approximations is given by

$$||\phi_1 - \phi_2||_{L^2(\Omega)} = \left(\int_\Omega \Big(\phi_1(\boldsymbol{x}) - \phi_2(\boldsymbol{x}) \Big)^2 \mathrm{d}V(\boldsymbol{x}) \right)^{1/2}.$$

If $\overline{\boldsymbol{\Phi}}_1$ and $\overline{\boldsymbol{\Phi}}_2$ were the same size then the L^2 error between ϕ_1 and ϕ_2 would be given by

$$||\phi_1 - \phi_2||_{L^2(\Omega)} = \Delta\mathcal{D} ||\overline{\boldsymbol{\Phi}}_1 - \overline{\boldsymbol{\Phi}}_2||_F$$

where

$$\Delta\mathcal{D} = \prod_{d=1}^{D} \Delta x_d.$$

However, $\overline{\boldsymbol{\Phi}}_2$ is exactly twice the size of $\overline{\boldsymbol{\Phi}}_1$. So, computing the L^2 error is not as straight forward, but we can use the fact that each control volume $\Omega(i_1, \ldots, i_D)$ contains 2^D of the control volumes $\widehat{\Omega}(j_1, \ldots, j_D)$. Define

$$\mathcal{J}(i_1, \ldots, i_D) := \{(j_1, \ldots, j_D) | j_d = 2i_d - 1 \text{ or } j_d = 2i_d\ \forall d \in \{1, \ldots, D\}\}$$

then

$$\mathrm{cl}\big(\Omega(i_1, \ldots, i_D)\big) = \bigcup_{j \in \mathcal{J}(i_1, \ldots, i_D)} \mathrm{cl}\Big(\widehat{\Omega}(j)\Big) \tag{A.5}$$

Now we can define an enlarged version of $\overline{\boldsymbol{\Phi}}_1$ by

$$\widehat{\boldsymbol{\Phi}}_1(j_1, \ldots, j_D) := \overline{\boldsymbol{\Phi}}_1(i_1, \ldots, i_D),\ \forall (j_1, \ldots, j_D) \in \mathcal{J}(i_1, \ldots, i_D),\ \forall i_1, \ldots, i_D.$$

This allows us to redefine ϕ_1 as

$$\begin{aligned} &\phi_1 : \Omega \to \mathbb{R} \\ &\phi_1(\boldsymbol{x}) = \widehat{\boldsymbol{\Phi}}_1(j_1, \ldots, j_D),\ \forall \boldsymbol{x} \in \widehat{\Omega}(j_1, \ldots, j_D),\ \forall j_1, \ldots, j_D. \end{aligned} \tag{A.6}$$

The L^2 between ϕ_1 and ϕ_2 is then given by

$$||\phi_1 - \phi_2||_{L^2(\Omega)} = 2^{-D}\Delta\mathcal{D}||\widehat{\boldsymbol{\Phi}}_1 - \overline{\boldsymbol{\Phi}}_2||_F.$$

The Kronecker product gives us an easy way to create the enlarged tensor $\widehat{\boldsymbol{\Phi}}_1(j_1, \ldots, j_D)$. We simply take

$$\widehat{\boldsymbol{\Phi}}_1(j_1, \ldots, j_D) = \overline{\boldsymbol{\Phi}}_1(j_1, \ldots, j_D) \otimes \mathbf{1}_D,$$

where $\mathbf{1}_D \in \mathbb{R}^{2\times\ldots\times 2}$ is a Dth-order tensor with all entries equal to 1. The TT-toolbox contains a function (`kron.m`) which extends Matlabs inbuilt Kronecker product function. This function computes the Kronecker product of two tensors in TT-format with the resultant tensor also being in TT-format. Therefore, we can quickly and efficiently compute the L^2 error between ϕ_1 and ϕ_2 by computing the Frobenius error between $\widehat{\boldsymbol{\Phi}}_1$ and $\overline{\boldsymbol{\Phi}}_2$. We summarise the results of this section in Proposition A.0.3.

Proposition A.0.3
Let ϕ solve a convection diffusion equation on some rectangular domain Ω in $\mathbb{R}^D$. Suppose that we have two finite volume approximations of ϕ at $t = T$ denoted by $\overline{\boldsymbol{\Phi}}_1 \in \mathbb{R}^{I_1\times\ldots\times I_D}$ and $\overline{\boldsymbol{\Phi}}_2 \in \mathbb{R}^{2I_1\times\ldots\times 2I_D}$ which are generated by the TT-FVM algorithm with spatial step-sizes $\Delta x_1, \ldots, \Delta x_D$ and $\Delta x_1/2, \ldots, \Delta x_D/2$ respectively. Then the L^2 error between the two approximations is given by

$$||\overline{\boldsymbol{\Phi}}_1 - \overline{\boldsymbol{\Phi}}_2||_{L^2} := 2^{-D}\Delta\mathcal{D}||\widehat{\boldsymbol{\Phi}}_1 - \overline{\boldsymbol{\Phi}}_2||_F$$

where

$$\widehat{\boldsymbol{\Phi}}_1(j_1, \ldots, j_D) := \overline{\boldsymbol{\Phi}}_1(j_1, \ldots, j_D) \otimes \mathbf{1}_D,$$

with $\mathbf{1}_D \in \mathbb{R}^{2\times\ldots\times 2}$ being a Dth-order tensor with all entries equal to 1.